Centro Universitário de Brasília – UniCEUB
Faculdade de Ciências Exatas e de Tecnologia – FAET
Curso de Engenharia da Computação
Disciplina: Projeto Final
Professor Orientador: Wladimir S. Meyer

TRANSMISSÃO ALTERNATIVA DE DADOS

Tiago Almeida Mitsuka
R.A.: 2001641/0

Brasília, 2º semestre de 2004

DEDICATÓRIA

Dedico este projeto aos meus pais, minhas irmãs e minha namorada, pelas angústias e preocupações que passaram por minha causa; pelo amor, carinho e estímulo que me ofereceram. Dedico-lhes essa conquista como gratidão.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus pela minha existência e pelas oportunidades que me foram concedidas em minha vida.

Ao meu mestre, professor Wladimir, que me ofereceu uma excelente orientação, transmitindo o verdadeiro valor deste trabalho.

RESUMO

O projeto mostra a implementação de um modelo de transmissão de dados simplex entre dois microcomputadores, sendo o dispositivo de saída do primeiro microcomputador o monitor de vídeo e o dispositivo de entrada do segundo microcomputador a porta serial de comunicação.

A transmissão é realizada por uma interface capaz de capturar sinais ópticos emitidos pelo monitor de vídeo e convertê-los em sinais elétricos. A transmissão é estabelecida entre os microcomputadores através dos *softwares* de transmissão e de recepção.

SUMÁRIO

ÍNDICE DE FIGURAS

LISTA DE TABELA

LISTA DE SIGLAS

API - Application Programming Interface
CD - Carrier Detect
CRT - Tubo Catódicos de Raios
CTS - Clear To Send
DAC - Conversor Analógico-Digital
DCE - Data Communications Equipment
DSR - Data Set Ready
DTE - Data Terminal Equipment
DTR - Data Terminal Ready
EEPROM - Electrically Erasable Programmable Read Only Memory
EIA - Eletronic Industries Association
MCU´s - Micro Controler Unit
PIC - Programmable Interrupt Controller
RAM - Random Access Memory
RISC - Reduced Instruction Set Computer
ROM - Read Only Memory
RTS - Request To Send
UART - Universal Asynchronous Receiver Transmitter
USART - Universal Synchronous /Asynchronous Receiver Transmitter
VGA - Video Graphics Array

1 INTRODUÇÃO

1.1 Contextualização do Projeto

As redes de computadores surgiram numa época em que a relação entre o usuário e o computador não trazia qualquer atrativo para estabelecer processos de ensino-aprendizagem por meio de suas interfaces, que eram cartões perfurados com códigos binários, encarregados de estabelecer o diálogo entre o homem e a máquina. Naquela época, meados dos anos 60, o interesse pelas redes de computadores estava centrado em estratégias de guerra, que demandavam informações cada vez mais volumosas a serem compartilhadas em tempo real por diferentes partes de um mesmo território.

Uma rede de computadores pode ser simplificadamente definida como a interligação física e lógica entre dois ou mais computadores. A interligação física se estabelece entre interfaces de comunicação, conhecidas como placas de rede ou placas de modulação-demodulação, também chamados de modens. As placas de rede são ligadas através de cabos, que são os meios físicos encarregados de transmitir os impulsos analógicos ou digitais entre os computadores. Esta parte é normalmente referida como hardware de comunicação. A parte lógica da interligação é feita pelos softwares de comunicação e envolve um conjunto de protocolos, especialmente desenvolvidos para este fim [1].

O primeiro acesso bidirecional ao mundo exterior oferecido por todos os computadores PCs era a porta de comunicações de dados assíncronos, também conhecida como porta assíncrona, porta serial ou porta de comunicações. Desde então, com o avanço da tecnologia, vem surgindo cada vez mais interfaces que permitem a troca de informação entre dispositivos, entre eles pode-se citar: porta serial, porta paralela, USB, Infravermelho, Bluetooth, Wireless entre outros.

1.2 Objetivo do Projeto

O projeto visa à implementação de um modelo didático de transmissão alternativa de dados e um estudo geral sobre o seu funcionamento. O modelo deve realizar uma transmissão de dados entre dois computadores, onde o dispositivo de saída do primeiro computador é o monitor de vídeo, e o dispositivo de entrada

utilizado pelo segundo computador é a porta serial de comunicação:

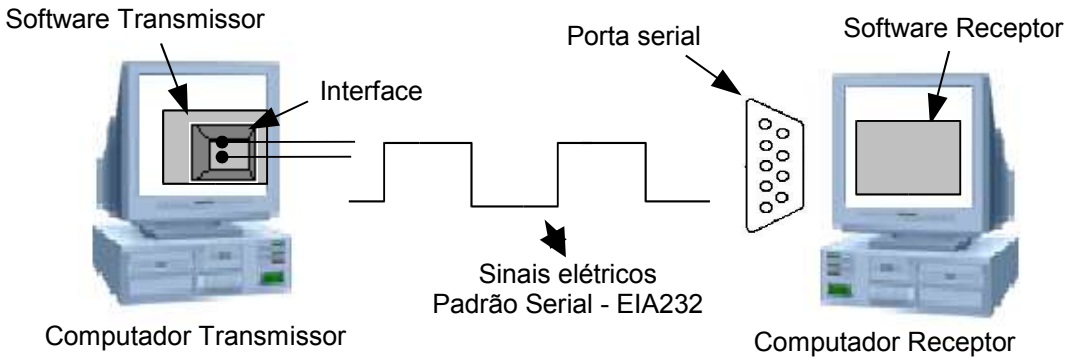


Figura 1.1 - Modelo da transmissão

A recepção dos dados é realizada através de uma interface capaz de capturar os dados do monitor do primeiro computador e enviá-los à porta serial do segundo computador.

Outros objetivos do projeto são: implementação do *Software Transmissor* e a implementação do *Software Receptor*, ambos fundamentais no estabelecimento do enlace de comunicação.

1.3 Motivação

A idéia inicial do projeto visava a implementação de uma interface de conversão entre sinais visuais emitidos pelo monitor de vídeo em sinais infravermelhos. Isso permitiria que qualquer dispositivo que contivesse um receptor infravermelho pudesse capturar dados do monitor de vídeo através da interface, evitando-se assim, a conexão física com qualquer outra porta de comunicação do PC.

Mas qual seriam os benefícios desse tipo de transmissão? Imagine que um usuário possua um telefone celular que contenha uma interface infravermelho. Agora imagine que esse usuário deseje receber um arquivo da Internet (como um toque polifônico, uma imagem ou até mesmo um jogo) em seu celular, através um computador público. Suponha também que, por questões de segurança, as interfaces desse computador, como porta serial, Infravermelho, porta paralela ou USB, estejam desabilitadas ou inacessíveis. Então, o usuário não poderia efetuar a transferência do arquivo para seu celular. Porém, com o uso da interface de conversão, o usuário poderia receber os dados em seu telefone celular tendo o

monitor de vídeo como transmissor.

Seria ainda muito mais interessante se, além do infravermelho, o celular já tivesse também essa interface acoplada em sua estrutura, o que permitiria a captura de dados do monitor de vídeo sem a necessidade de qualquer porta de comunicação.

Por questões de facilidade, e por ser um padrão amplamente adotado nos PCs, foi escolhido o padrão serial assíncrono, para transmissão dos dados visuais convertidos pela interface, ao invés de ser usada a transmissão por infravermelho.

Outro motivo preponderante que levou à escolha do padrão serial, foi a existência de muitos microcontroladores no mercado que já possuem uma UART (Universal Asynchronous Receiver Transmitter) interna, permitindo a transmissão de dados no padrão adotado.

1.4 Estrutura do trabalho

Além desse capítulo introdutório, este trabalho está estruturado com mais 5 capítulos, assim distribuídos:

O capítulo 2 explora o funcionamento do monitor de vídeo e as características que permite o seu uso como transmissor de dados. O capítulo aborda também o funcionamento do sistema de recepção e a comunicação de dados com o microcomputador.

O capítulo 3 mostra como o protocolo de comunicação está estruturado, os tipos de transmissão que podem ser utilizados pela interface de comunicação e suas vantagens.

O capítulo 4 aborda principalmente o funcionamento e a tecnologia utilizada na implementação dos *softwares* de transmissão e recepção de dados.

O capítulo 5 aborda o funcionamento do protótipo e os principais componentes utilizados na sua construção.

Por fim, a conclusão traz as considerações finais sobre o trabalho, as principais contribuições, os resultados obtidos, as dificuldades encontradas e as sugestões para trabalhos futuros.

2 CARACTERÍSTICAS DO ENLACE DE COMUNICAÇÕES

2.1 O monitor como transmissor de dados

Um sistema de transmissão completa inclui geralmente: um transmissor, um meio de transmissão através do qual a informação é transmitida; e um receptor, que produz uma réplica identificável da informação de entrada. Na maioria dos sistemas, a transmissão de informação é estreitamente relacionada com a modulação, isto é, consiste na colocação de dados digitais num sinal analógico com a variação de um determinado sinal denominado portadora. Como fenômeno físico que é, um sinal possui diversas grandezas físicas mensuráveis. Se o emissor produzir variações nestas grandezas de modo a traduzir a informação a transmitir, então o receptor pode detectar estas variações e obter a informação que foi transmitida [2].

A transmissão de dados sobre sinais analógicos justifica-se pela necessidade de aproveitar algumas infra-estruturas analógicas, no caso desse projeto, a estrutura aproveitada é a característica responsável pela geração de imagens do monitor, sendo utilizada para transmitir sinais ópticos.

Portanto, no sistema de comunicação de dados utilizado no projeto, a placa de vídeo juntamente com o monitor de vídeo serão os responsáveis pela modulação do sinal analógico utilizado na transmissão de dados.

Para entender como o sinal pode ser modulado, é necessário primeiramente entender como o ambiente operacional ou o programa aplicativo gera as imagens formadas na tela do monitor, o processo é descrito da seguinte maneira:

- Os sinais digitais do ambiente operacional ou do programa aplicativo são recebidos pela placa de vídeo;

- A placa submete o sinal a um circuito denominado *conversor analógico-digital* (DAC). Em geral, o circuito DAC está no interior de um microcircuito especializado, que na realidade contém três DACs – um para cada cor primária usada no monitor: vermelho, azul e verde.

- O DAC compara os valores digitais enviados pelo PC com uma *tabela de busca* que contém os valores de ajuste de níveis de tensão para as três cores primárias necessárias para criar a cor de um único pixel. Em uma placa VGA (Video Graphics Array) normal, a tabela contém

valores para 262,144 cores possíveis, das quais 256 valores podem ser armazenados na memória da placa VGA simultaneamente. As placas Super VGA têm memória suficiente para armazenar 16 bits de informação para cada pixel (16.000 cores, denominadas *high color*), ou 24 bits por pixel (16.777.216 – ou *true color*);

- A placa envia os sinais aos três canhões eletrônicos localizados na parte posterior do *tubo de raios catódicos* (CRT) do monitor. No vácuo existente no interior do CRT, cada canhão eletrônico dispara um feixe de elétrons, um para cada uma das três cores primárias. A intensidade de cada feixe é controlada pelos sinais da placa [9].

A Figura 2.1 mostra o processo geração de imagem no monitor descrito acima.

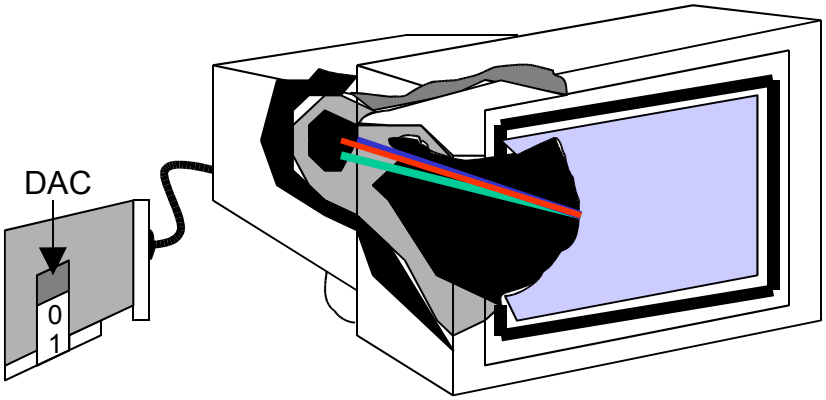


Figura 2.1 - Funcionamento do monitor de vídeo.

O que permite a modulação do sinal de transmissão a partir do monitor como transmissor de dados é a possibilidade de manipulação da intensidade dos feixes de elétrons pelo sistema a partir de um aplicativo, no caso desse projeto, foi implementado um *software* específico para essa função.

A imagem da tela é formada pelos três feixes eletrônicos independentes que caminham em conjunto, sendo que, um deles é responsável pela formação do vermelho, o outro pelo verde e outro pelo azul. Os feixes percorrem a tela continuamente, da esquerda para a direita, de cima para baixo, fazendo seu percurso formando linhas horizontais. Ao chegar na parte direita da tela, o feixe é apagado momentaneamente e surge novamente na lateral esquerda da tela, mas posicionando um ponto mais abaixo, e percorre novamente a tela da esquerda para a direita, formando outra linha. Este processo se repete até que o feixe chegue à

parte inferior da tela. O feixe é então apagado momentaneamente e surge novamente na parte superior da tela, pronto para percorrê-la novamente. A Figura 2.2 mostra como o feixe triplo de elétrons varre toda tela do monitor indicando o sincronismo vertical e horizontal.

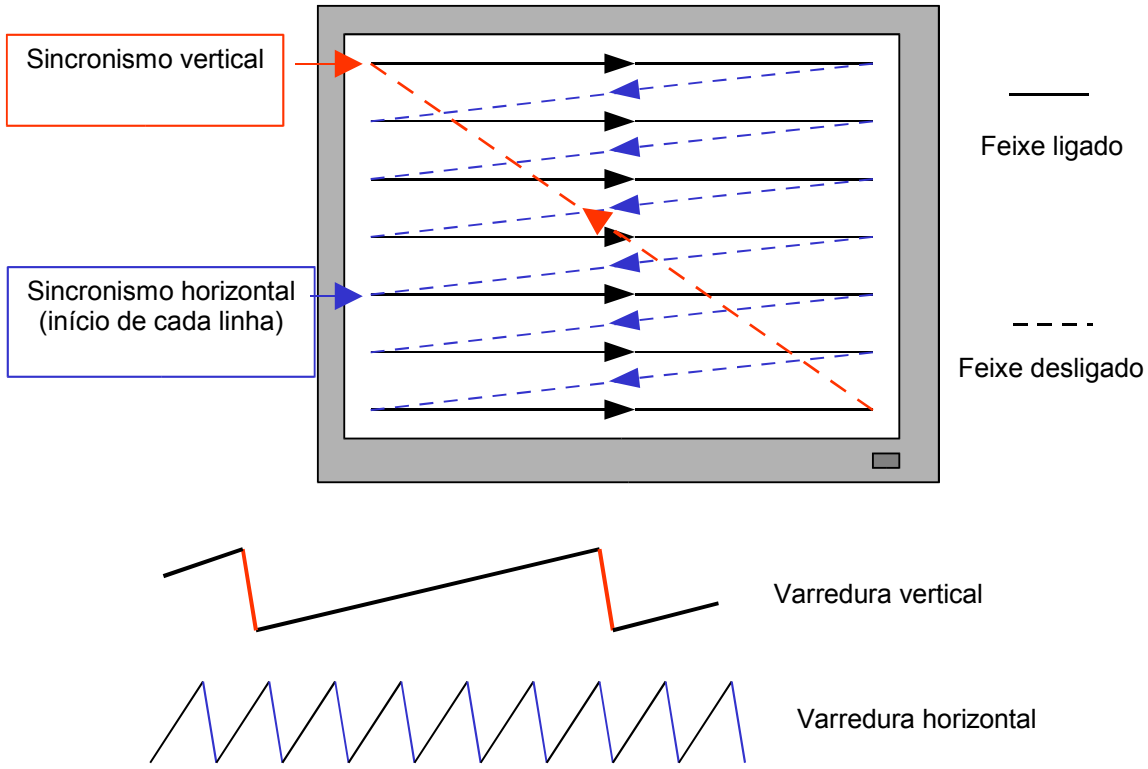


Figura 2.2 - Percurso realizado pelo feixe triplo de elétrons.

A velocidade desse feixe é muito alta. Na maioria dos monitores modernos, o feixe eletrônico descreve mais de 50.000 linhas por segundo. Em termos técnicos, isto é o mesmo que dizer que o monitor está operando com uma freqüência horizontal de 50kHz.

Em uma resolução de 800x600 a trajetória do feixe triplo forma 600 linhas. Na resolução de 640x480 são percorridas 480 linhas. Seja qual for o caso, o número de linhas descritas pelo feixe é igual à resolução vertical.

O tempo que leva o feixe eletrônico, ao chegar no fim da tela, para ser movido da parte inferior até a parte superior da tela, é chamada de *retraço vertical*. Esse período demora cerca de 5% a 10% do período necessário para o feixe descrever todas as linhas da tela.

Então, utilizando uma freqüência horizontal de 50 kHz e uma resolução de

800x600, calcula-se a taxa de atualização da seguinte maneira:

Levando em conta 600 linhas, com a demora de 5% a 10% de retraço vertical, calcula-se um total de 660 linhas, considerando-se o valor máximo de retraço vertical. Como o feixe eletrônico percorre 50.000 linhas por segundo, o número de vezes que este feixe percorre a tela inteira em um segundo será igual a 50.000 / 600, que é igual a 75, ou 75Hz. Isso significa que a taxa de atualização desse monitor é de 75 Hz.

Em função da freqüência vertical e do número de linhas descritas pelo feixe, pode-se calcular o número de vezes que a tela é preenchida a cada segundo. O recomendável é que a tela inteira seja preenchida por cerca de 75 vezes por segundo, valores muito baixos podem causar desconforto visual [8].

Hipoteticamente, se o olho humano enxergasse a passagem do feixe eletrônico varrendo a tela de um monitor, operando com uma freqüência vertical de 60 Hz, observaria que o feixe passa por um único ponto, ou pixel, 60 vezes por segundo, o que corresponderia a uma freqüência de 60 Hz. Portanto, o feixe estimula eletricamente o mesmo ponto (composto de fósforo) de acordo com a freqüência vertical.

Na transmissão analógica a informação é transmitida por um sinal chamado portadora, fazendo com que esta portadora varie proporcionalmente com o sinal ou a informação que se quer transmitir. Utilizando o monitor como transmissor de dados, a freqüência da portadora de transmissão é igual à freqüência vertical do monitor. Aumentando a taxa de atualização do monitor, aumenta-se também o número de vezes que o feixe passa por um único ponto por segundo, conseqüentemente aumenta-se também a velocidade do sinal da portadora.

Um sistema analógico em que a informação é enviada pela variação proporcional da amplitude da portadora, recebe o nome de modulação em amplitude. Esse tipo de modulação pode ser utilizado para transmitir dados através do monitor de vídeo.

A tela do monitor é formada por um grande número de pontos minúsculos, e cada ponto do monitor é formado por um composto de fósforo que possui uma propriedade interessante – emite luz quando atingido pelo feixe de elétrons. Utilizando essa propriedade para transmitir dados e utilizando apenas um ponto da tela como fonte de transmissão óptica, conclui-se que: variando a intensidade do

feixe de elétrons que atinge um ponto, resulta na variação da luminosidade emitida por ele, o que corresponde à variação da amplitude do sinal da portadora.

Portanto, o monitor de vídeo pode ser utilizado como transmissor de dados através da emissão de sinais visuais, sinais que posteriormente serão transformados em sinais elétricos. O sinal visual de transmissão emitido pelo monitor possui a freqüência da portadora determinada pela freqüência vertical do monitor e sua modulação realizada pela variação da intensidade com que o feixe eletrônico incide sobre a tela do monitor (variação da luminosidade emitida pela tela). Porém, a comunicação só pode ser realizada em uma direção (transmissão simplex), do monitor para o módulo receptor.

2.2 Módulo receptor

Para transformar em sinais elétricos as informações que se apresentam originalmente na forma de sons ou imagens, os sistemas de comunicação utilizam transdutores eletroacústicos, eletromecânicos ou optoeletrônicos. O fotodetector, utilizado para transformar luz em sinais elétricos, é um exemplo de transdutor optoelétrico utilizado no projeto [2].

O fotodetector tem um papel importante na transmissão: fazer a tradução do sinal óptico gerado pelo monitor de vídeo em um sinal elétrico para posteriormente ser transmitido para o receptor da mensagem. O fotodetector, mais especificamente o fotodiodo, possui uma sensibilidade suficiente para capturar o momento em que o feixe eletrônico passa por um ponto na tela do monitor.

A interface projetada pode operar com uma freqüência máxima igual à freqüência vertical do monitor de vídeo. A figura abaixo mostra um sinal elétrico convertido por um fotodiodo de um monitor operando com uma freqüência vertical de 60 Hz.

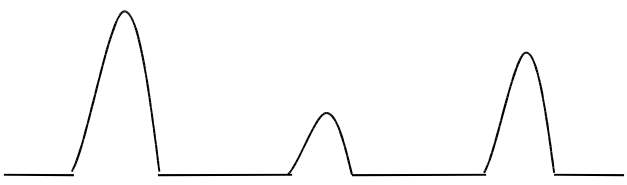


Figura 2.3 - Sinal elétrico gerado por um transdutor óptico.

Sabendo-se que o sinal representado na Figura 2.3 representa a transformação de um sinal óptico de um monitor operando a uma freqüência vertical de 60Hz, calcula-se o período do sinal da portadora da seguinte maneira:

Como:

$$f = 60\,\text{Hz} \text{ e } f = \frac{1}{P}$$

Tem-se:

$$60\text{Hz} = \frac{1}{P} \Rightarrow P = \frac{1}{60\text{Hz}}$$

$$P = 16{,}6\,\text{ms}$$

Portanto, conclui-se que, como explicado no capítulo anterior, a freqüência vertical do monitor de vídeo é igual à freqüência do sinal da portadora de transmissão, conseqüentemente ambos, possuem os mesmos períodos de sinal.

Para transformar os sinais provenientes de transdutores em sinais adequados para a transmissão por meio de ondas eletromagnéticas, utilizam-se circuitos moduladores, que são essenciais nos transmissores.

Para recuperar a informação incorporada ao sinal modulado, os receptores utilizam circuitos demoduladores, que operam seguindo os mesmos princípios utilizados nos moduladores (pelo menos quando se trata de sinais modulados em amplitude). Para a demodulação, é necessário aplicar o sinal modulado e a portadora em um demodulador, para que na saída obtenha-se o sinal modulante [2].

Ainda na Figura 2.3 observa-se que o sinal possui amplitudes diferentes, isso se deve ao fato de que o feixe, que passa pelo ponto de captura do fotodiodo, varia a sua intensidade a cada atualização ou *refresh* na tela do monitor. O controle dessa variação permite que o sinal seja modulado de acordo com a regra de transmissão.

A estrutura básica do sistema de recepção do sinal utilizado no projeto é semelhante ao sistema utilizado na recepção por fibra óptica ilustrada na figura abaixo:

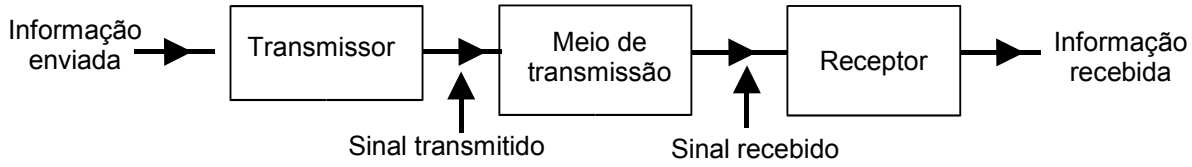


Figura 2.4 - Estrutura básica de um sistema de transmissão.

O fotodetector transforma sinais ópticos em elétricos, que são em seguida processados por circuitos eletrônicos, restaurando a informação original. Contudo, o projeto consiste em uma interface de transmissão, sendo necessário um passo a mais na estrutura do sistema de recepção, que consiste na transmissão dos sinais elétricos (dados) para o equipamento receptor, e a forma escolhida foi à transmissão serial assíncrona, padrão EIA232.

2.3 Comunicação com o microcomputador

Apesar da variedade de nomes dada à porta serial, todas as portas são idênticas, ao menos no funcionamento. Elas recebem palavras de cinco (ou mais bits) de cada vez e as transformam numa cadeia de pulsos em "fila indiana". Como os bits de informações são transferidos como uma longa série de pulsos, essa forma de transmissão de dados é chamada serial. A única propriedade de uma corrente de dados seriais é que os dados são transmitidos e recebidos como um único sinal, por um único pino de um dispositivo:

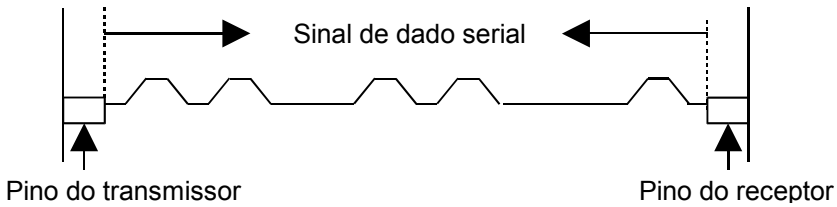


Figura 2.5 - Transmissão serial.

Na transmissão serial um único circuito é suficiente para transferir um sinal de um ponto a outro, porém, o computador não oferece nenhuma garantia que lerá

sempre o primeiro bit de cada série quando começar a acompanhar a marcha dos sinais seriais; basta um erro nesse momento para que todos os bytes subseqüentes sejam interpretados equivocadamente.

Um dos métodos para solucionar esse problema é a transmissão assíncrona. A alternativa é acrescentar marcas aos *strings* de dados, indicando o início e o fim de cada bloco de dados. O sistema receptor poderá, então, identificar o início de cada série e evitar erros de interpretação sem que precise recorrer, para isso, a um sinal de sincronização. Os sinais assíncronos formam a base dos sistemas operacionais.

Na maioria dos sistemas assíncronos, os dados são divididos em pequenos grupos, cada um dos quais correspondem aproximadamente a um byte. Esses grupos de dados são chamados de *palavras*, e podem ter entre cinco e oito bits de dados. Os tamanhos de palavras mais comuns têm sete e oito bits: o primeiro, porque é suficiente para comportar todos os caracteres ASCII maiúsculos e minúsculos; o segundo, porque cada palavra corresponde exatamente a um byte de dados.

Um pulso muito especial de comprimento duplo denominado *start bit* (bit de início, ou bit de partida) é acrescentado a esses dados, e indica o início de uma palavra de dados. Um *stop bit* (bit de fim, ou bit de término) indica o fim da palavra. Entre o último bit de uma palavra e o *stop bit*, há geralmente um bit de *paridade* para verificação da integridade dos dados [9].

Juntos, os bits de dados, o *start bit* o bit de paridade e o *stop bit* compõem um frame (quadro) de dados.

Em 1960, um comitê de normatização, hoje conhecido como EIA (Eletronic Industries Association), desenvolveu uma interface padrão para equipamentos de comunicação de dados. O padrão RS232 nasceu da necessidade de assegurar a confiabilidade da comunicação e permitir a interconexão de equipamentos produzidos por diferentes fabricantes. Ele especifica o sinal de voltagem, tempo de sinal, função do sinal, um protocolo de troca de informações e os conectores mecânicos.

Desde que o padrão foi criado, a EIA publicou três modificações, a mais recente é o padrão EIA232E, introduzida em 1991. Além da mudança de nome de RS232 para EIA232, algumas linhas de sinais foram renomeadas e várias novidades

definidas, incluindo uma blindagem para o condutor.

O padrão EIA232 especifica um conector de 25 pinos (DB-25) para conexão com dispositivos serias e uma versão reduzida, com nove pinos (DB-9). Ambos os conectores são usados pelas portas seriais dos PCs. A especificação dos sinais dos pinos DB-9, é ilustrado na figura abaixo.

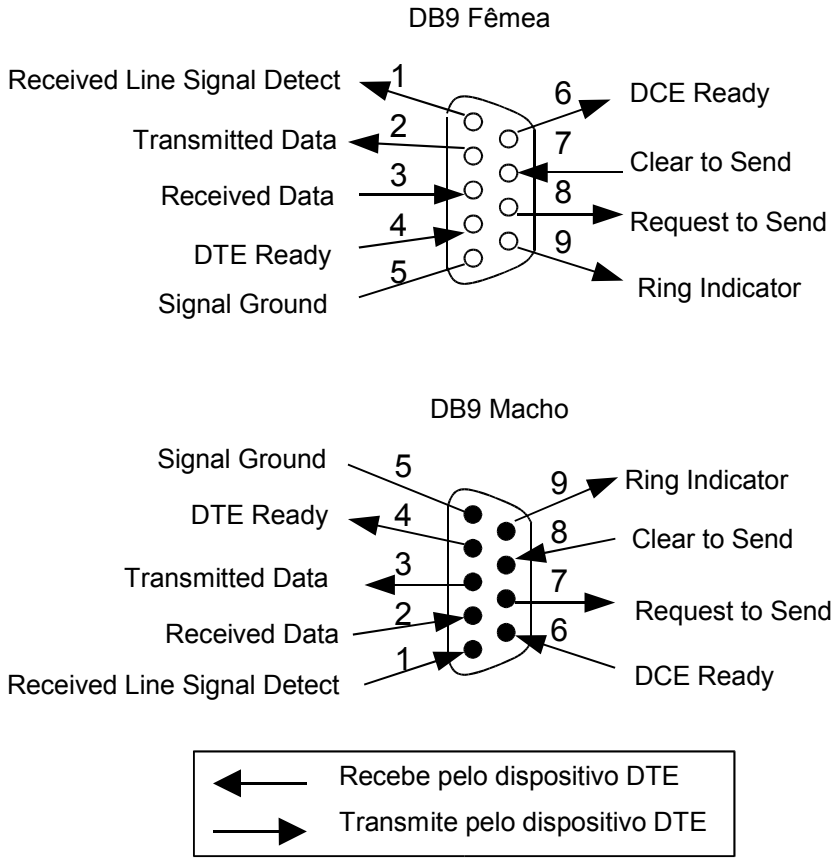


Figura 2.6 - Definição dos sinais do conector DB-9 *macho* e *fêmea*.

Os nomes dos sinais que implicam em uma direção, como “Transmit Data” e “Receive Data”, são nomeados do ponto de vista dos dispositivos DTE (Data Terminal Equipment). Se a norma EIA232 for seguida à risca, estes sinais terão o mesmo nome e o mesmo número de pino do lado do DCE (Data Communications Equipment). Infelizmente, isto não é feito na prática pela maioria dos engenheiros, provavelmente porque em alguns casos torna-se difícil definir quem é o DTE e quem é o DCE.

A tabela a seguir apresenta a convenção utilizada para os sinais mais comuns do conector DB-9:

Tabela 2.1 - Pinos e funções dos sinais do conector DB-9

Pino	Nome	Descrição
1	Carrier Detect (CD)	Também chamado de Data Carrier Detect (DCD). Este sinal é relevante quando o DCE for um modem. Ele é habilitado (nível lógico “0”) quando a linha telefônica está “fora do gancho”, uma conexão for estabelecida, e um tom de resposta começar a ser recebido do modem remoto. Este sinal é desabilitado (nível lógico “1”) quando não houver tom de resposta sendo recebido, ou quando o tom de resposta for de qualidade inadequada para o modem local.
2	Received Data (RxD)	Este sinal está ativo quando o DTE receber dados do DCE. Quando o DCE estiver em repouso, o sinal é mantido na condição de marca (nível lógico “1”, tensão negativa).
3	Transmitted Data (TxD)	Este sinal está ativo quando dados estiverem sendo transmitidos do DTE para o DCE. Quando nenhum dado estiver sendo transmitido, o sinal é mantido na condição de marca (nível lógico “1”, tensão negativa).
4	DTE Ready (DTR)	Também chamado de Data Terminal Ready. Este sinal é habilitado (nível lógico “0”) pelo DTE quando for necessário abri o canal de comunicação. Se o DCE for um modem, a habilitação do sinal DTR prepara o modem para ser conectado ao circuito do telefone, e uma vez conectado, mantém a conexão. Quando o sinal DTR for desabilitado (nível lógico “1”), o modem muda para a condição “no gancho” e termina a conexão.
5	Ground (GND)	Sinal de terra utilizado como referência para outros sinais.
6	DCE Ready (DSR)	Também chamado de Data Set Ready. Quando originado de um modem, este sinal é habilitado (nível lógico “0”) quando as seguintes forem satisfeitas: 1 – O modem estiver conectado a uma linha telefônica ativa e “fora do gancho”; 2 – O modem estiver no modo dados; 3 – O modem tiver completado a discagem e está gerando um tom de resposta. Se a linha for tirada do gancho, uma condição de falha for detectada, ou uma conexão de voz for estabelecida, o sinal DSR é desabilitado (nível lógico “1”).
7	Request To Send (RTS)	Este sinal é habilitado (nível lógico “0”) para preparar o DCE para aceitar dados transmitidos pelo DTE. Esta preparação inclui a habilitação dos circuitos de recepção, ou a seleção a direção do canal em aplicações half-duplex. Quando o DCE estiver pronto, ele responde habilitando o sinal CTS.
8	Clear To Send (CTS)	Este sinal é habilitado (nível lógico “0”) pelo DCE para informar ao DTE que a transmissão pode começar. Os sinais RTS e CTS são comumente utilizados no controle do fluxo de dados em dispositivos DCE.
9	Ring Indicator (RI)	Este sinal é relevante quando o DCE for um modem, e é habilitado (nível lógico “0”) quando um sinal de chamada estiver sendo recebido na linha telefônica. A habilitação desse sinal terá aproximadamente a duração do tom de chamada, e será desabilitado entre os tons ou quando não houver tom de chamada presente.

O padrão EIA232 também especifica o uso de nível de tensão para representar os valores “0” e “1”. Sinais com tensão entre –3 volts e –25 volts com relação ao terra são considerados nível lógico “1” (condição marca), e tensões entre +3 volts e +25 volts são considerados nível lógico “0” (condição espaço). A faixa de tensões entre –3 volts e +3 volts é considerada uma região de transição para o qual o estado do sinal é indefinido.

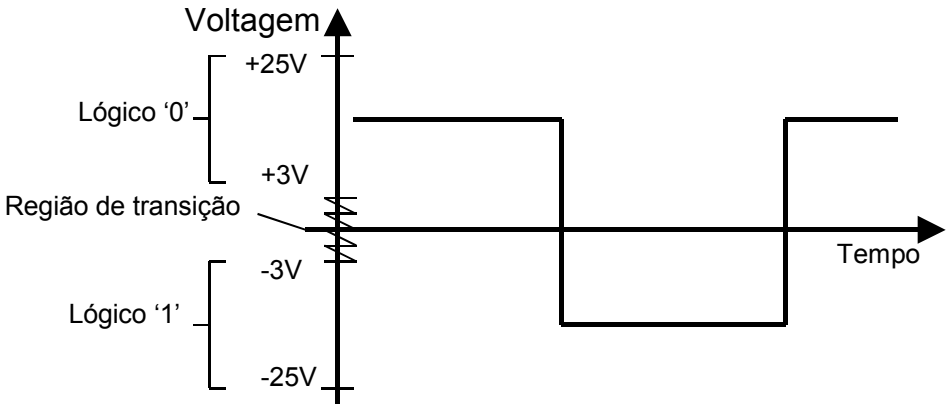


Figura 2.7 - Nível de tensão no padrão EIA232.

O chip capaz de transmitir e receber dados no formato assíncrono é chamado de UART. O propósito da UART é converter dados para bits, enviá-los pela linha serial e reconstruir os dados recebidos.

Neste projeto, um microcontrolador PIC (Programmable Interrupt Controller) é o responsável pelo tratamento dos sinais recebidos pelo fotodetector e pelo envio dos dados para a porta serial do computador receptor. O que viabiliza o uso do microcontrolador é que o mesmo já possui uma USART (Universal Synchronous/Asynchronous Receiver Transmitter) interna, assim como uma série de dispositivos embutidos que facilitam a implementação do *hardware*.

Para eliminar grande parte das preocupações decorrentes do uso de uma conexão serial, como problemas de fiação e controle de fluxo por hardware, é utilizado o cabo “*null modem*”. Usando voltagens locais, pode-se fazer com que uma porta serial acredite estar recebendo os sinais esperados de uma conexão remota. Por exemplo, é possível utilizar a voltagem positiva que o próprio PC transmite na forma de um sinal DTR de modo a fazer com que a porta serial acredite estar recebendo o DSR, o CTS e o CD, através da ligação dos quatro fios entre si dentro do conector serial.

Na Figura 2.8 é apresentado um modo de conexão de um cabo “null modem”. Apenas 3 fios são necessários (TxD, RxD e GND), porém, no caso do projeto, a transmissão é simplex, necessitando apenas de dois fios. A teoria de operação é razoavelmente simples. O princípio é fazer o DTE detectar a conexão com um modem. Qualquer dado transmitido pelo microcontrolador deve ser recebido pelo DTE. O sinal de terra (SG) também deve ser conectados ao terra comum aos dois dispositivos.

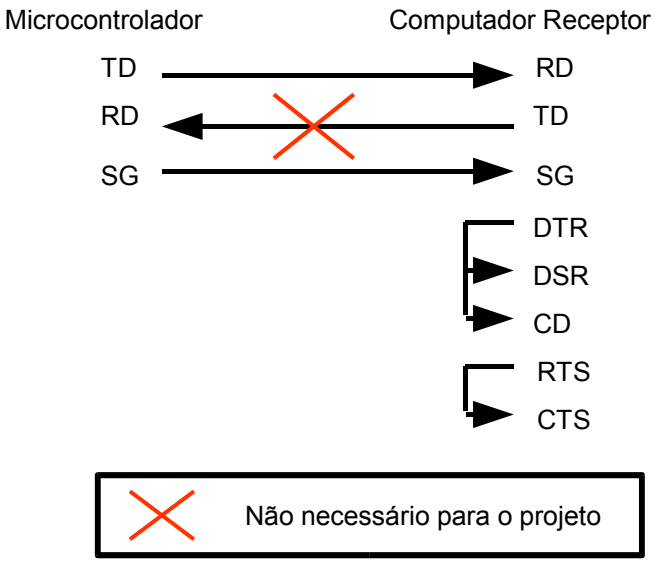


Figura 2.8 - Ligação do cabo *null modem*.

O sinal DTR é conectado com os sinais DSR e CD nos dois extremos. Quando o sinal DTR for ativado (indicando que o canal de comunicação está aberto), imediatamente os sinais DSR e CD são ativados. Nessa hora o DTE detecta que o Modem Virtual ao qual está conectado está pronto e que foi detectado uma portadora no outro modem. O DTE trata agora dos sinais RTS e CTS. Como os 2 dispositivos se comunicam à mesma velocidade, o fluxo de controle não é necessário e conseqüentemente essas 2 linhas são conectadas juntas no DTE. Quando o computador está preparado, ele ativa a linha RTS, como estão conectadas juntas, imediatamente recebe a resposta que o outro DTE está pronto pela linha CTS.

Nota-se que o sinal RI não está conectado em nenhum extremo. Esta linha é utilizada apenas para informar ao DTE que existe um sinal de chamada telefônica presente. Como não existe modem conectado a linha telefônica ela pode permanecer desconectada.

Portanto, o cabo *null modem* juntamente com um conversor de nível serão os principais responsáveis pela transmissão entre o microcontrolador e a porta serial do computador receptor dos dados. O funcionamento do conversor de nível será explicado mais adiante.

3 O PROTOCOLO DE COMUNICAÇÃO

O protocolo de comunicação foi projetado de forma a prover uma comunicação entre o monitor e a interface serial assíncrona de forma mais simples possível em uma transmissão simplex. A impossibilidade de uma transmissão half-duplex ou full-duplex tem um motivo muito óbvio: o monitor é um dispositivo de saída, e sua principal função é exibir as informações processadas ao usuário, não permitindo a troca de informações.

Com o objetivo de elaborar um protocolo simples, não foram implementados nenhum tipo de sinalização e nenhum sistema de detecção e controle de erro entre o monitor de vídeo e a interface, porém, um bit de *start* e um bit de *stop* são posteriormente inseridos pela USART na comunicação serial entre a interface e o computador receptor.

A freqüência do sinal de transmissão é igual à freqüência vertical do monitor de vídeo em operação, como atualmente a maioria dos monitores estão sendo fabricados com uma faixa de freqüência vertical que varia entre 50Hz e 160Hz, a freqüência da portadora de transmissão adotada é de 60Hz, pois a maioria dos monitores aceitam essa freqüência. Como conseqüência disso, em uma transmissão binária (adotando duas grandezas para a transmissão dos níveis lógicos “0” e “1”), a taxa de transmissão é de 60 bits por segundo ou 60 bps.

Todos os bits capturados do monitor são utilizados como dados. Os dados são enviados pelo *Software Transmissor* com o tamanho de palavras de oito bits, enviados em seqüência. Para maximizar a quantidade de bits de informações transmitidas não é utilizado nenhum bit exclusivo de sincronização de dados, ao invés disso, o próprio bit transmitido é utilizado para o sincronismo. Para possibilitar esse sincronismo, os níveis de tensão (grandezas física utilizadas para representar os dígitos binários "uns" e "zeros") são diferentes de zero volt, isso permite que tanto o bit 1 quanto o bit 0 sejam detectados pelo circuito.

Os sinais com amplitude entre 1,6 volts e 3,3 volts são detectados pelo circuito como nível lógico “0”, e amplitudes entre 3,3 volts e 5 volts são detectados como nível lógico “1”. Os sinais com amplitude entre 0 volts e 1,6 volts não são detectados pelo circuito. A Figura 3.1 representa os níveis de tensão e o valor do bit representado por eles.

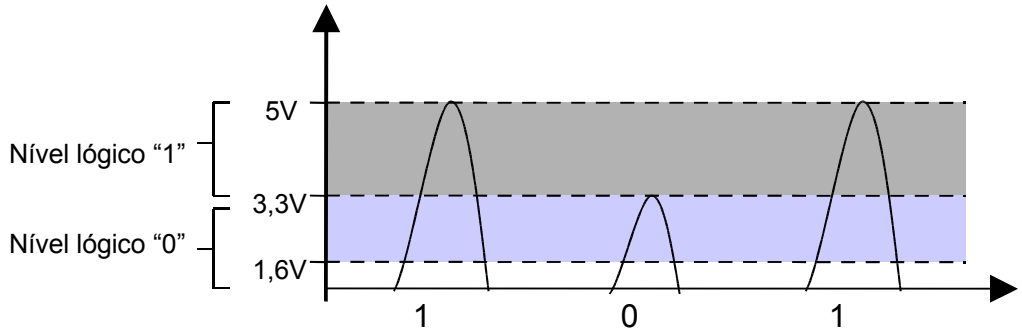


Figura 3.1 - Representação dos níveis de tensão dos sinais binários.

No modelo implementado, o sentido da transmissão é do monitor para uma porta serial, porém a velocidade mínima de transmissão aceita pela porta serial é de 110 bauds, e a taxa de transmissão adotada é de 60 bps. A solução encontrada foi à utilização de um microcontrolador PIC para gerenciar a incompatibilidade de velocidade de transmissão.

O microcontrolador foi programado para receber as informações transmitidas pelo monitor e armazená-las em uma variável, quando a informação atinge um tamanho de 8 bits ela é transmitida por uma UART no padrão serial para o computador receptor.

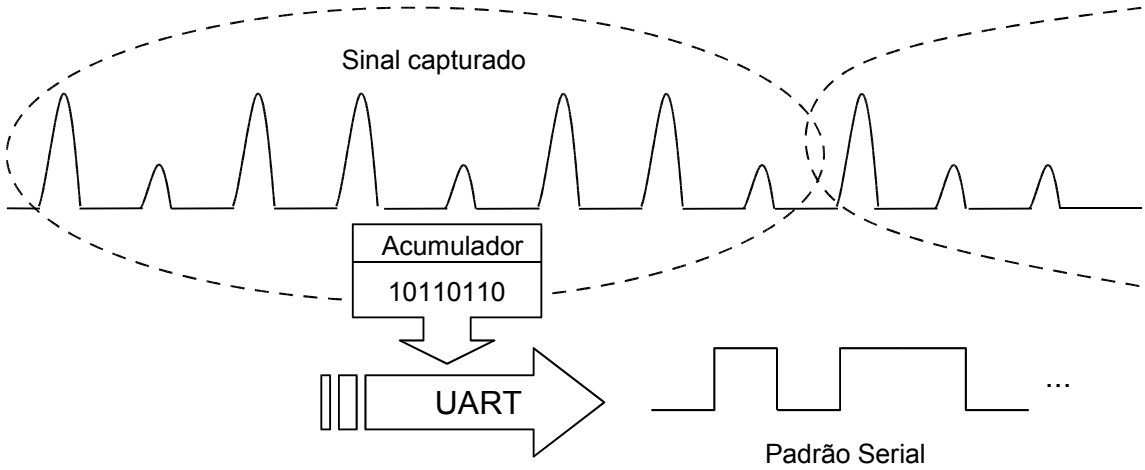


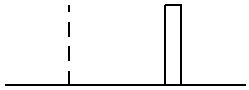
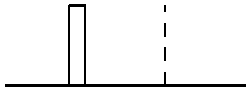
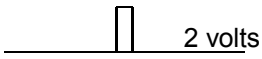
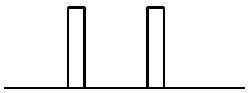
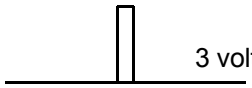
Figura 3.2 - Tratamento de dados realizado pela interface.

Portanto, o microcontrolador aguarda o recebimento de todos os bits antes de enviá-los a porta serial do computador receptor. A UART tem que ser rápida o suficiente para enviar os dados no intervalo entre o oitavo bit (informação completa para envio) e o próximo bit, pois o microcontrolador deve estar preparado para

armazenar o bit seguinte antes de sua chegada.

Uma das soluções encontradas para aumentar a velocidade de transmissão é a utilização de uma transmissão em multi-níveis. Para entender como essa transferência funciona considere o caso de 4 mensagens equiprováveis. Se essas mensagens são transmitidas por pulsos binários, é preciso um grupo de dois pulsos binários por mensagens. Cada pulso binário pode assumir dois estados, por isso uma combinação de dois pulsos forma 4 padrões distintos, que são associados a uma das quatro mensagens (Tabela 3.1). Portanto, precisa-se de dois pulsos binários para transmitir qualquer uma das quatro mensagens equiprováveis. Cada uma dessas mensagens toma duas vezes o tempo necessário para transmitir uma de duas mensagens equiprováveis e, por isso, contém duas vezes mais informações, ou seja, 2 bits.

Tabela 3.1 - Equivalências.

Símbolos	Equivalente com dígitos binários	Forma de onda com pulsos binários	Equivalentes com dígitos quaternários	Forma de onda com pulsos quaternários
A	00		0	
B	01		1	
C	10		2	
D	11		3	

A quantidade de informação contida em qualquer uma de *n* mensagens equiprováveis é igual a $\log_2 n$ bits. Isso implica que um mínimo de $\log_2 n$ bits pulsos binários são necessários para transmitir essa mensagem.

Supondo que haja uma necessidade de transmitir qualquer uma das quatro mensagens equiprováveis, em vez de duas. Obviamente, não é possível transmitir essa informação por um único pulso binário, uma vez que ele pode assumir apenas dois estados. Porém, podemos transmitir qualquer uma das quatro mensagens por

um grupo de dois pulsos binários. Cada pulso binário pode assumir dois estados, e por isso, uma combinação de dois pulsos formará quatro padrões distintos, como mostra a Tabela 3.1. O estado zero de um pulso (ausência de um pulso) é mostrado por uma linha tracejada. Portanto, precisa-se de dois pulsos binários para transmitir qualquer uma das quatro mensagens equiprováveis. Por isso, a informação transmitida por mensagem de dois bits.

De outro modo, pode-se transmitir essa informação por um pulso quaternário, que pode assumir quatro estados ou quatro níveis, por exemplo, 0, 1, 2 e 3 volts. Cada estado corresponde a um dos quatro símbolos possíveis. Qualquer um dos quatro níveis possíveis pode ser transmitido por um único pulso quaternário. Segue-se que um único pulso quaternário pode transmitir a informação carregada por dois pulsos binários e, por isso, carregar 2 bits de informação. É fácil ver que em um pulso, que pode assumir M estados ou M níveis distintos, carrega uma informação de $\log_2 M$ bits.

Portanto, quanto maior o número de níveis distintos que um pulso pode assumir, maior a informação contida em cada pulso. Uma vez que, em qualquer canal prático, há sempre uma certa quantidade de sinal ruído, será impossível distinguir, no receptor, níveis cujas separações sejam menores que a amplitude do sinal de ruído. Por isso, a consideração do ruído exige que os níveis estejam separados, pelo menos, da amplitude do ruído [2].

Por exemplo, adotando o modelo de transmissão pelos sinais ópticos do monitor, a variação da intensidade ou amplitude do sinal em 4 níveis distintos possibilita a transmissão de dois bits em um único sinal da portadora de transmissão:

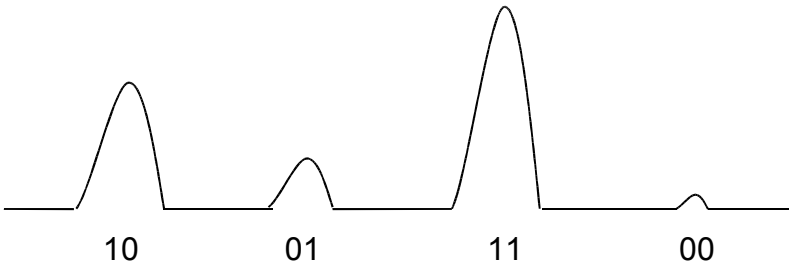


Figura 3.3 - Transmissão multi-nível.

Portanto, uma das formas de aumentar a velocidade de transmissão do modelo implementado é utilizando a transmissão multi-nível. Contudo, para melhor entendimento de qual a diferença, em termos de velocidade de transmissão, entre as duas técnicas abordadas, discute-se a seguir, as duas grandezas muito utilizadas

em transmissão de dados: a *velocidade de modulação* e a *velocidade de transmissão*.

A velocidade de modulação, também conhecida como taxa de modulação, indica quantas vezes por segundo um sinal é transmitido. A unidade de velocidade de modulação é baud, definida por:

$$Vm = \frac{1}{d}(\text{baud})$$

Onde:
Vm = velocidade de modulação.
d = largura da menor transição do sinal digital.

Exemplo: Um sinal digital possui uma duração de 16,66 ms. Determine sua velocidade de modulação.

Solução:

$$Vm = \frac{1}{16{,}66} = 60 \text{ bauds}$$

A velocidade de transmissão indica a quantidade de bits transmitidos por segundo. A velocidade de transmissão é proporcional à velocidade de modulação e ao número de bits enviados em cada transição do sinal. O número de bits, N, depende do número de níveis do sinal. Um sinal de dois níveis contém um bit de informação. É chamado de sinal binário. Assim, um sinal de dois níveis transmitidos 60 vezes por segundo proporcionará uma velocidade de transmissão de 60 bps. A seguinte equação relaciona a velocidade de transmissão, Vt, com a velocidade de modulação, Vm:

$$Vt = VmxN$$

Onde:
Vt = velocidade de transmissão, em bps (bits por segundo);
Vm = velocidade de modulação, em bauds;
N = número de bits associados a cada nível do sinal digital.

Existe uma relação entre o número de níveis, M, de um sinal e seu número de bits associado. Essa relação é dada pela seguinte equação:

$$M = \frac{\log N}{\log 2}$$

Exemplo: Calcular o número de bits associados a cada nível de um sinal de quatro níveis, como o representado na Figura 3.4.

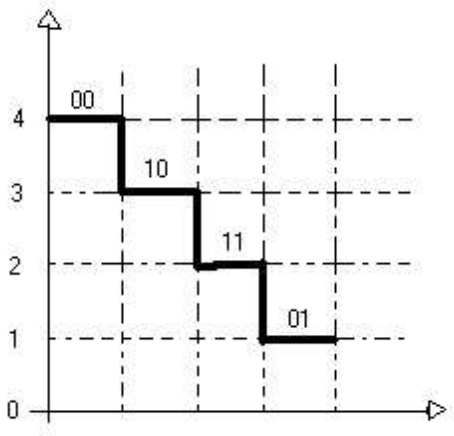


Figura 3.4 - Sinal digital de quatro níveis, com codificação *gray*.

Solução:

$$\mathrm{M} = \frac{\log \mathrm{N}}{\log 2} = \frac{\log 4}{\log 2} = 2$$

Embora a transmissão em multi-nível seja mais rápida que a transmissão binária, o projeto enfocará somente na última, adotando apenas dois níveis de representação, o que permite uma transmissão de 60 bps. Os fatores determinantes para escolha da técnica de transmissão adotada é a necessidade de um maior aprofundamento na técnica de modulação do sinal recebido pelo monitor e pelo fato de que a transmissão em dois níveis é menos sensível a perturbações externas ou diversidade de monitores. Essa diversidade pode causar diferenças de intensidade de brilho entre os monitores, o que acarretaria sinais de portadora com amplitudes distintas, necessitando assim, de um ajuste no circuito para modular o sinal em cada monitor.

4 ROTINAS DE TRATAMENTO DE DADOS

4.1 Transmissão

O *Software Transmissor* é o responsável em transformar os dados a serem enviados em sinais ópticos exibidos na tela do monitor. Sinais que posteriormente serão capturados pela interface e enviados no padrão serial para o *Software Receptor*.

O *software* basicamente converte um dado ou um byte, em um valor binário, em outras palavras, ele *quebra* o byte em bits. Esses bits são exibidos da seguinte forma:

- Se o bit representa um nível lógico “1”, a tela do monitor é preenchida com uma intensidade de cor que possa ser entendida pela interface como nível alto (geralmente é utilizada a cor branca);
- Se o bit representa um nível lógico “0”, a tela do monitor é preenchida com uma intensidade de cor que possa ser entendida pela interface como nível baixo (geralmente é utilizada a cor cinza).

Portanto, se a tela do monitor é preenchida com uma cor de intensidade muito baixa (cores escuras como o preto), a interface não entende como sendo um dado (bit), não transmitindo nada. Para todas as outras intensidades de cores podem ser entendidas ou como nível lógico “0” ou como nível lógico “1”.

O *Software Transmissor* é o item mais crítico do projeto, pois o seu funcionamento correto influencia toda performance do sistema. O software tem que ser capaz de redesenhar o dado na tela (bit a ser transmitido) na mesma velocidade que a taxa de atualização do monitor (freqüência vertical). Por exemplo, suponha que o software esteja programado para enviar uma seqüência de quatro bits. Se o software não estiver sincronizado com a taxa de atualização do monitor, e a sua velocidade de redesenho for igual a metade da velocidade de atualização da tela, então, quando o software estiver transmitindo um bit, a tela é atualizada duas vezes. Isso significa que, quando for transmitido os bits ‘1100’, a interface irá capturar os bits dobrados: ‘11110000’. No caso inverso, se a velocidade de redesenho do software for igual ao dobro da velocidade de atualização do monitor, então, quando o software estiver transmitido dois bits, a tela é atualizada apenas uma vez. Isso

significa que, quando for transmitido os bits ‘1100’, a interface irá capturar apenas os bits ‘10’.

O *Software Transmissor* inicialmente foi implementado em Java, porém, o mesmo não ofereceu uma performance satisfatória. A solução encontrada foi dividir o *software* em dois programa que interagem entre si:

O programa principal, implementado em Java, consiste no programa de *front-end,* que oferece uma interface de configuração gráfica para o usuário. O programa principal permite a configuração da intensidade (cor) em que a tela será preenchida, permite também, ao usuário, digitar a seqüência de caracteres que será transmitida pelo *Software Transmissor*.

A figura abaixo mostra o *Software Transmissor* configurado para transmitir a seqüência de caracteres: “Transmitindo dados - PROJETO FINAL”.

Figura 4.1 - *Software Transmissor*.

O programa secundário foi implementado em C++ utilizando a API Directx. O programa apenas transmite os dados de acordo com os parâmetros recebidos pelo programa principal.

Um exemplo é a transmissão do caractere “a”. Em binário, ele é representado na tabela ASCII como sendo ‘01100001’. Então, o *Software Transmissor* preenche a tela na seguinte seqüência: cinza, branco, branco, cinza, cinza, cinza, cinza e branco. Com isso, na outra ponta da transmissão, o *Software Receptor* receberá o caractere “a” como resposta a transmissão.

Qualquer programa, passa por toda “burocracia” do sistema operacional. Essa “burocracia” diminui a performance do programa. Para resolver esse problema, o software implementado utiliza um método que permite que o programa desenhe diretamente na tela do monitor, tornando-o mais eficiente. A técnica utilizada é o modo de tela cheia exclusiva, que permite também o acesso às funções de configurações de vídeo, tais como: resolução da imagem, número de bits por pixel e taxa de atualização do monitor.

Supondo que o programa desenhe pixel por pixel ou linha por linha na tela do monitor, o mesmo não teria performance suficiente para fazê-lo na mesma velocidade em que a tela é atualizada pelo monitor. A solução encontrada é a utilização de uma técnica muito aplicada por programadores de jogos, chamada *double-buffering* (armazenamento duplo).

A superfície da tela é normalmente conhecida como *superfície primária* e a imagem não exibida, usada pelo *double-buffering,* é normalmente conhecida como *back buffer*. Portanto, o *double-buffering* trata-se de uma técnica que permite a manipulação de uma imagem no *back buffer* enquanto a *superfície primária* está sendo exibida na tela.

Uma superfície primária é usualmente manipulada através de um objeto gráfico, uma operação de manipulação direta à memória da tela. O ato de copiar o conteúdo de uma superfície para outra é conhecido como *blitting* ou simplesmente *blt*, como representado na figura abaixo:

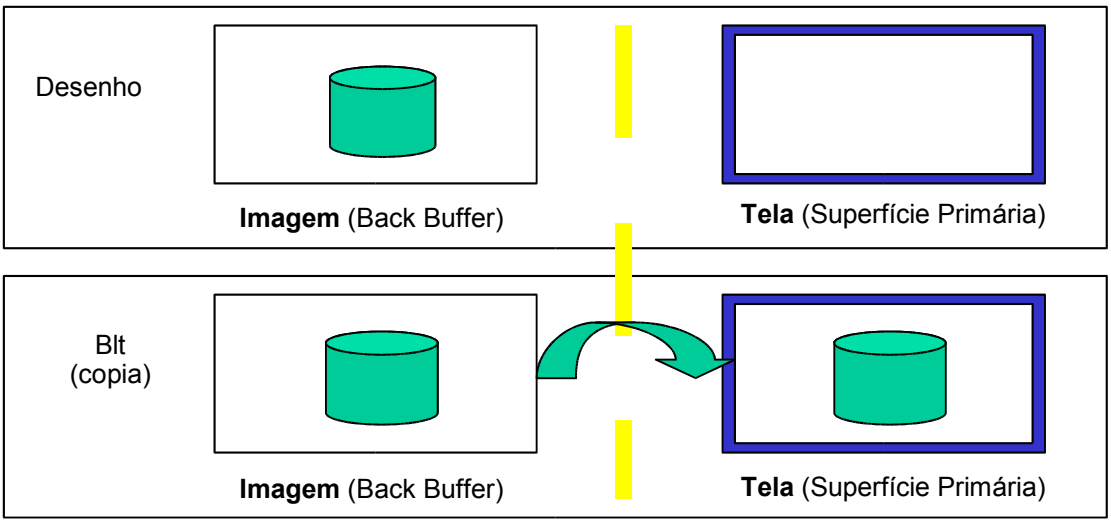


Figura 4.2 - Técnica de *Double-Buffering.*

Outra técnica que pode ser utilizada com o modo de tela cheia exclusiva é o *page-flipping*, que também é uma forma de *double-buffering*.

Muitas placas gráficas têm a noção de ponteiro de vídeo, que é simplesmente um endereço na memória de vídeo. Este ponteiro diz para a placa gráfica onde ela deve procurar pelo conteúdo de vídeo a ser exibido no próximo ciclo de *refresh.* Isso permite que um desenho criado no *back buffer* seja exibido no próximo ciclo, ao invés de ser copiado para a superfície primária.

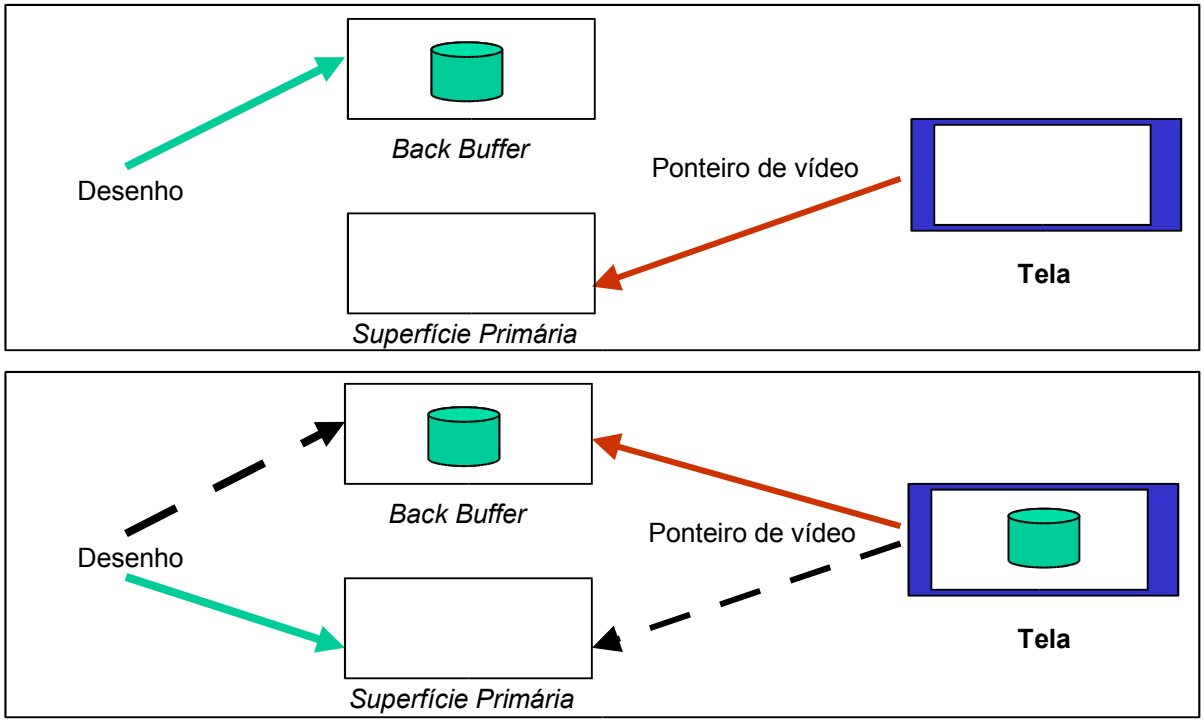


Figura 4.3 - Técnica de *Page-Fipping*.

Em algumas situações, é mais vantajosa a utilização de múltiplos back fuffers. Esta técnica é utilizada principalmente quando o tempo de desenho é maior que o tempo de *refresh*.

Como a técnica de *page-flipping* é mais rápida, ela é utilizada na implementação do *Software Transmissor*. Os códigos fonte do programa principal e do programa secundário encontram-se no **ANEXO B**.

4.2 Recepção

A função do *Software Receptor* é bastante simples, ele apenas captura os dados transmitidos pela interface de conversão na porta serial e os exibe ao usuário. O software deve ser configurado para operar com os mesmos parâmetros de transmissão definidos no microcontrolador.

O *Software Receptor* ao ser executado exibe uma janela que permite ao usuário escolher a porta de comunicação em que os dados serão recebidos.

Figura 4.4 - Janela de configuração.

O software permite também uma opção de visualização dos dados em caracteres no padrão ASCII ou em binário (dígitos “uns” e “zeros”). O código fonte do *Software Receptor* encontra-se no **ANEXO D**.

A figura abaixo mostra o *Software Receptor* recebendo uma seqüência de caracteres na porta de comunicação COM1.

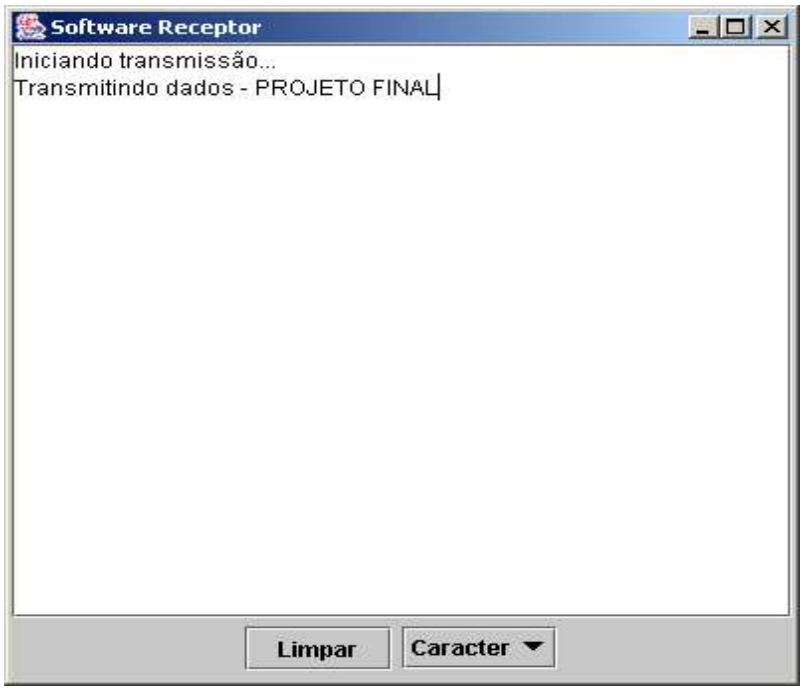


Figura 4.5 - *Software Receptor*.

O *Software Receptor* foi implementado na linguagem Java, utilizando a API (Application Programming Interface) “Java Communications”. A SUN fornece como *download* gratuito a API de comunicação serial e paralela na URL: http://java.sun.com/products/javacomm/index.jsp.

5 CONSTRUÇÃO DO PROTÓTIPO

5.1 Estrutura Geral

A interface de transmissão funciona basicamente como um conversor de sinal analógico para digital, convertendo os sinais ópticos emitidos pelo monitor de vídeo em sinais digitais (nível lógico “0” e nível lógico “1”). Posteriormente, esses sinais convertidos são transmitidos para o dispositivo receptor no padrão serial EIA232. Portanto, a interface de transmissão divide-se em três módulos descritos pelo diagrama abaixo:

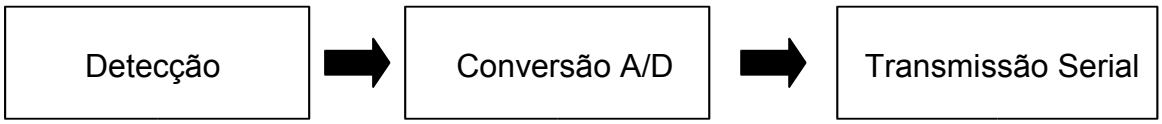


Figura 5.1 - Diagrama de blocos do transmissor.

O módulo de detecção é realizado por um transdutor óptico, enquanto a conversão dos sinais A/D e a transmissão no padrão serial são realizadas por um único dispositivo, um microcontrolador PIC.

Os MCU´s (Micro Controler Unit) ou microcontroladores, ao contrário dos microprocessadores, são dispositivos mais simples, com memórias RAM (Random Access Memory) e ROM (Read Only Memory) internas, oscilador interno de clock, I/O interno, entre outros, sendo por isso chamados muitas vezes de computadores em um único chip. Tais características tornam mais simples o projeto de dispositivos inteligentes, pois os MCU´s raramente necessitam de CI´s externos para funcionar, o que contribui para a diminuição de custos e tamanho.

Os microcontroladores PIC são uma família de dispositivos fabricados pela Microship. Utilizando uma arquitetura RISC (Reduced Instruction Set Computer), com freqüência de clock de até 40Mhz, até 2024k word de memória de programa e etá 3968 bytes de memória RAM. Além disso, podem ser encontrados com diversos periféricos internos, como: até quatro temporizadores/contadores, memória EEPROM (Electrically Erasable Programmable Read Only Memory) interna, gerador/comparador/amostrador PWM, conversores A/D de até 12 bits, interface de barramento CAN, I2C, SPI, entre outros [5].

O dispositivo utilizado na construção da interface é o PIC 16F628A. Este dispositivo é membro de uma família intermediária (Mid-range) da Microchip.

Suas principais características são:

- Baixo custo;
- Facilidade de programação;
- Grande diversidade de periféricos internos;
- Memória de programa do tipo FLASH;
- Excelente velocidade de execução.

Além disso, podem-se também destacar as seguintes especificações:

- 1024 x 14 bits de memória FLASH;
- 224 x 8 bits de memória SRAM disponíveis para o usuário;
- 128 x 8 bits de memória EEPROM interna;
- Pilha com 8 níveis;
- 15 pinos de I/O (entrada ou saída);
- 1 pino de entrada;
- 1 timer/contador de 8 bits;
- 1 timer/contador de 16 bits;
- 1 timer de 8 bits;
- 1 canal PWM com captura e amostragem (CCP);
- 1 canal de comunicação USART serial;
- 2 comparadores analógicos com referência interna programável de tensão;
- 1 timer watchdog;

- 10 fontes de interrupção independentes;
- Capacidade de corrente de 25 mA por pino de I/O;
- 35 instruções;
- Freqüência de operação de desde DC (0 Hz) até 20 Mhz;
- Oscilador 4Mhz/37Kh interno;
- Tensão de operação entre 3.0 a 5.5V;
- Compatível pino a pino com 16F84 e outros PICs de 18 pinos.

A presença de todos estes dispositivos em um espaço extremamente pequeno, dá ao projetista ampla gama de trabalho e enorme vantagem em usar um sistema microprocessado, onde em pouco tempo e com poucos componentes externos podemos fazer o que seria trabalhoso fazer com circuitos tradicionais. Foram esses e outros motivos que levaram a escolha do microcontrolador PIC.

A seguir, a figura abaixo mostra a pinagem do PIC 16F628A:

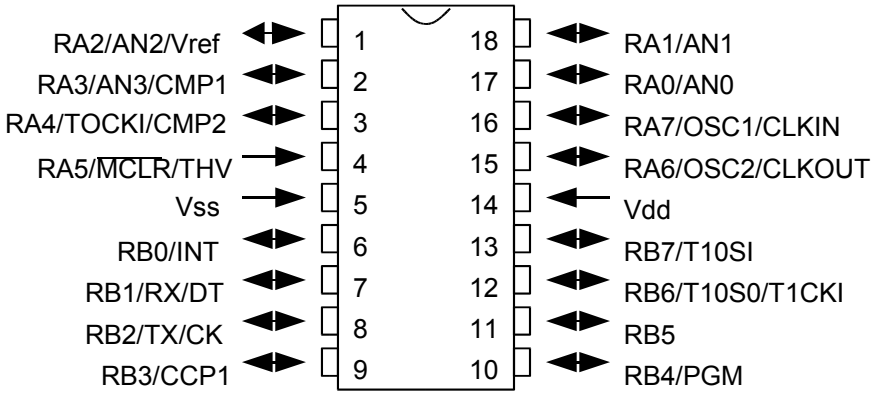


Figura 5.2 - Pinagem do PIC 16F628A.

Como explicado anteriormente, não será implementada no protótipo a transmissão em multi-nível. Para esse tipo de implementação seria necessária a utilização de um conversor A/D. A adição desse periférico pode ser feita ou pela troca do microcontrolador PIC 16F628A por um mais robusto como o 16F877A, que já contém um conversor interno A/D de 10 bits, ou pela adição externa de um conversor A/D ao circuito.

Atualmente, a maioria dos microcontroladores disponíveis no mercado conta com compiladores de linguagem C para o desenvolvimento de software. O uso de C permite a construção de programas e aplicativos muito mais complexos do que se fosse usado apenas o *Assembly*.

Além disso, o desenvolvimento em C permite uma grande velocidade na criação de novos projetos, devido às facilidades de programação oferecidas pela linguagem e também a sua portabilidade, o que permite adaptar programas de um sistema para outro com um mínimo de esforço.

Outro aspecto favorável da utilização da linguagem C é a sua eficiência. Eficiência no jargão dos compiladores é a medida do grau de inteligência com que o compilador traduz um programa em C para o código de máquina. Quanto menor e mais rápido o código gerado, maior será a eficiência da linguagem e do compilador. A linguagem C, devido a sua proximidade com o hardware e o *Assembly*, é uma linguagem extremamente eficiente. De fato, C é considerada como a linguagem de alto nível mais eficiente atualmente disponível.

Além disso, a utilização de uma linguagem de alto nível como C permite que o programador preocupe-se mais com a programação da aplicação em si, já que o compilador assume para si tarefas como o controle e localização das variáveis, operações matemáticas e lógicas, verificação de banco de dados de memória, etc [5].

De um modo geral, o primeiro passo para a construção do protótipo é uma alimentação correta, pois é de uma grande importância para o bom funcionamento do sistema de um microcontrolador. Pode comparar-se este sistema a um homem que precisa respirar. É provável que um homem que respire ar puro viva mais tempo que um que viva num ambiente poluído.

Para que o circuito funcione convenientemente, é necessário usar uma fonte de alimentação estável. O regulador de tensão de 3 conexões é um dispositivo comumente utilizado na regulagem de tensão. Pode-se imaginá-lo como um tipo especial de zener. Este dispositivo apresenta três conexões (entrada, saída e terra) e são regulados na fábrica para uma saída fixa de tensão (positiva para a família 78xx e negativa para 79xx). A função do regulador é manter constante a tensão, garantindo que a tensão seja limitada a um valor preestabelecido.

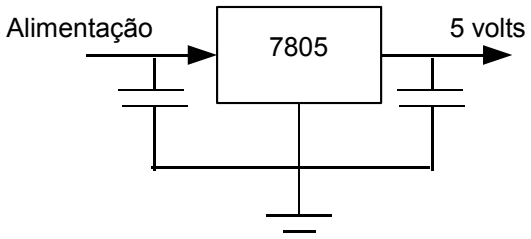


Figura 5.3 - Circuito esquemático da alimentação.

Pequenos capacitores são colocados em paralelo com a entrada e a saída do regulador, para protegê-lo de algum possível ruído elétrico e estabilizar a saída sob certas circunstâncias.

A necessidade de utilização do regulador de tensão foi constatada após vários testes com resultados indesejáveis, mesmo com boas fontes de tensão as interferências externas atrapalham o funcionamento correto do circuito.

5.2 Módulo de detecção

O componente responsável pela transdução dos sinais ópticos (dados luminosos) enviados pelo monitor de vídeo em um sinal elétrico analógico é o fotodiodo, que nada mais é que um diodo semicondutor em que a junção é exposta à luz. O fotodiodo é construído de modo a possibilitar a utilização da luz como fator determinante no controle da corrente elétrica que passa através dele. A energia luminosa desloca elétrons para a banda de condução, reduzindo a barreira de potencial pelo aumento do número de elétrons, que podem circular, se aplicada polarização reversa.

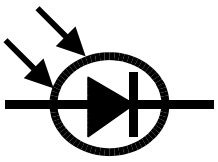


Figura 5.4 - Símbolo esquemático do fotodiodo.

O fotodiodo funciona da seguinte forma: se for polarizado inversamente na presença de luz ele permite a passagem de corrente elétrica e produz uma queda de tensão; se for polarizada reversamente na ausência de luz ele só permite a passagem de uma corrente muito pequena (praticamente nula) chamada de corrente reversa. Portanto, um fotodiodo ideal funcionaria como uma chave comum

que fecha quando há presença de luz e abre quando há ausência de luz. A figura abaixo resume a idéia de chave.

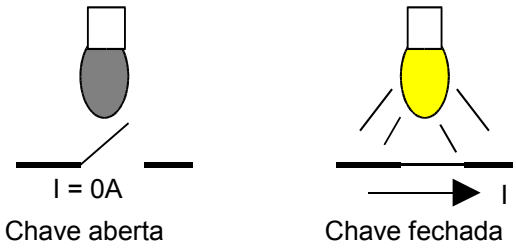


Figura 5.5 - Funcionamento de um fotodiodo ideal.

O fotodiodo é um componente bastante sensível. Nos testes realizados foi possível observar com o auxílio de um osciloscópio que o fotodiodo não detecta somente a luminosidade emitida pela tela do monitor como um todo, mas captura cada passagem do feixe de elétrons em um ponto da tela.

O circuito de detecção foi projetado de forma que, a cada estímulo luminoso, a corrente reversa que passa no fotodiodo passe também em uma resistência colocada em série com o mesmo, gerando assim uma tensão sobre a resistência. O circuito faz uma transdução de corrente para tensão:

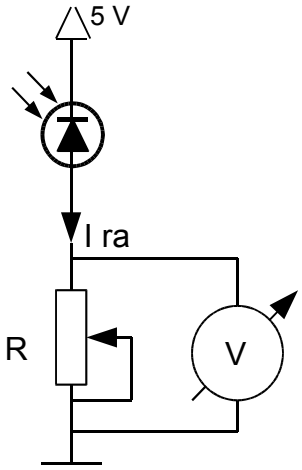


Figura 5.6 - Circuito esquemático do transdutor.

Em vez de ser colocada uma resistência em série com o fotodiodo foi colocado um potenciômetro de forma a permitir o ajuste do valor de pico na tensão do sinal analógico. Esse ajuste é necessário, pois as entradas analógicas do microcontrolador não podem ultrapassar o valor de alimentação do próprio CI (5 volts), e nem o valor do sinal analógico pode ser muito baixo, senão o nível lógico “1” não pode ser detectado pelo comparador de tensão (a função do comparador será detalhada no próximo item).

5.3 Módulo de conversão A/D

O próximo módulo a ser apresentado é o módulo de conversão A/D. Após a detecção do sinal da portadora, o mesmo é convertido de sinal analógico para digital. Como o protótipo implementa apenas uma transmissão binária, envolvendo somente dois valores de transmissão (nível lógico “0” e nível lógico “1”), não é utilizado nenhum dispositivo conversor A/D, e sim dois comparadores analógicos acoplados no próprio microcontrolador PIC. O funcionamento do comparador é detalhado no **ANEXO A**.

O primeiro comparador é ajustado para detectar a presença de todos os bits, independente de seu valor binário, funcionando como um gerador de sinais de sincronismo. O segundo, é ajustado de forma a detectar se o sinal da portadora representa um nível lógico “0” ou “1”.

O módulo comparador no microcontrolador é configurado para que os dois comparadores (C1 e C2) compartilhem a mesma referência externa (entrada não-inversora do comparador) e ambas as saídas encontrem-se disponíveis em pinos externos:

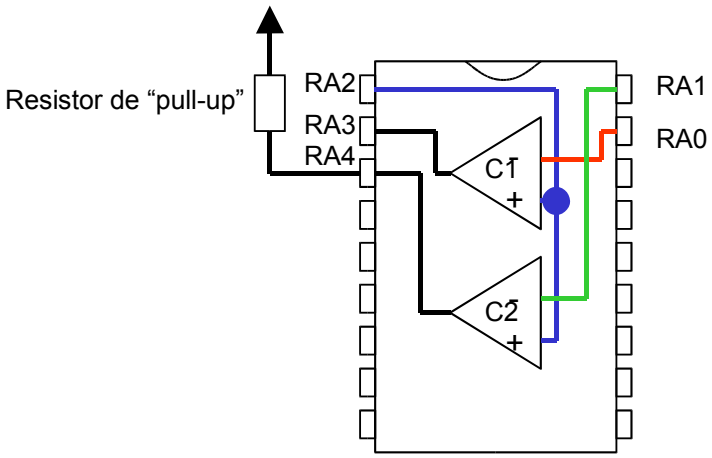


Figura 5.7 - Configuração dos comparadores C1 e C2.

Somente o comparador C2 possui um dreno aberto, portanto é necessário a utilização de um resistor de *pull-up* para seu funcionamento normal. No **ANEXO A**, encontra-se a explicação do por quê da utilização do resistor no comparador.

Os valores de tensão de referência externa nas entradas inversoras dos comparadores são ajustados respectivamente para 2/3 e 1/3 do valor da tensão de pico da portadora do sinal capturado, e a entrada não-inversora de ambos os

comparadores recebe o sinal capturado. Como a amplitude máxima do sinal não pode ultrapassar a 5 volts, são utilizados os valores aproximadamente de 3,3 volt e 1,6 volt. Os valores de tensão de referência são ajustados por meio de divisores de tensão, de acordo com a expressão abaixo:

$$V1 = \frac{R2}{R1 + R2}.V = \frac{5K\Omega}{10K\Omega + 5K\Omega}.5V = 1{,}66V$$

$$V2 = \frac{R1}{R1 + R2}.V = \frac{10K\Omega}{10K\Omega + 5K\Omega}.5V = 3{,}33V$$

A Figura 5.8 mostra a configuração completa dos comparadores de tensão utilizando os valores calculados na expressão acima.

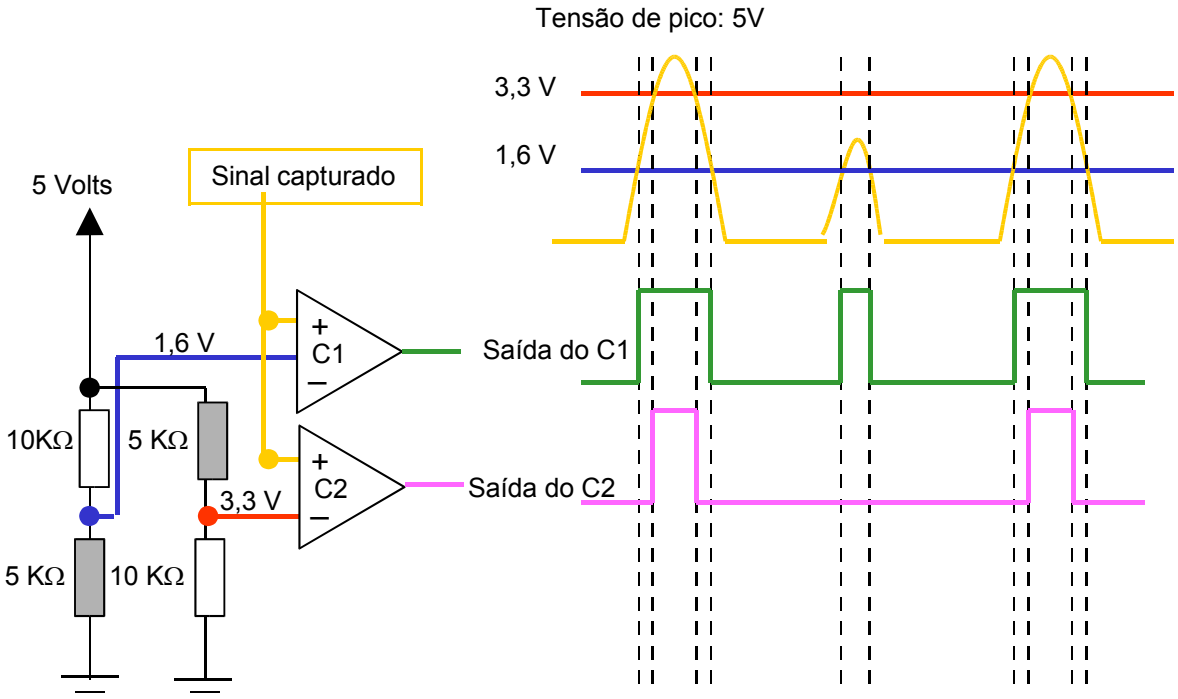


Figura 5.8 - Configuração das referências externas.

Portanto, o comparador C1 é usado para detectar a presença de todos os bits recebidos pelo sinal, e o comparador C2 detectar apenas a presença do nível lógico “1”. Porém, não basta apenas que os comparadores gerem as saídas de tensão, para que o programa em execução no microcontrolador possa capturar os sinais é necessário um evento de interrupção.

A interrupção é um evento externo ao programa que provoca parada da sua execução, a verificação e tratamento do referido evento e, em seguida, o retorno do

programa ao ponto em que havia sido interrompido. As estruturas de interrupção são utilizadas para que a CPU tome conhecimento de eventos de alta prioridade para o programa.

Quando ocorre um evento de interrupção, a instrução em execução é completada, o conteúdo do contador de programa (ou equivalente) é armazenado na pilha e o programa é desviado para um endereço conhecido como vetor de interrupção.

No vetor de interrupção, deve existir uma sub-rotina que deverá providenciar o devido tratamento da interrupção e em seguida fazer o retorno da interrupção (o que normalmente é feito por uma instrução de retorno específica), o que conduz o programa de volta ao seu fluxo normal [4].

No microcontrolador são utilizados dois eventos de interrupção, cada um é acionado pelo sinal de saída de cada um dos comparadores internos.

O módulo comparador possui uma interrupção própria, porém o uso da mesma é inviável devido às interrupções serem compartilhadas entre os dois comparadores, e também porque a transição de nível de borda em que interrupção é acionada não pode ser configurada, sendo acionada tanto na borda de subida como na borda de descida.

Portanto, os dois comparadores geram a *chamada* de uma única interrupção e sem o controle de transição de nível de borda. Dessa forma, seriam geradas interrupções em excesso (quatro interrupções em vez de duas) em um mesmo método de tratamento de dados.

Utilizando uma interrupção externa e um evento de captura o problema de interrupções em excesso é resolvido, porque dessa forma, é possível separar uma interrupção na borda de subida e uma interrupção de borda de descida, uma interrupção para cada comparador, com métodos distintos.

A interrupção externa é uma forma básica de interrupção disponível em qualquer microcontrolador ou microprocessador. Trata-se basicamente de um flip-flop que tem a saída ativada pelo evento de interrupção e desativada pelo programa do usuário. Nos PICs, pode-se selecionar a borda de sensibilidade da interrupção externa (se ela será ativada numa transição de nível “0” para nível “1” ou uma transição de nível “1” para nível “0” do sinal externo).

O modo de captura funciona um pouco diferente da interrupção externa, sua principal função é determinar o período de um sinal aplicado à entrada CCP1, sendo o resultado armazenado em um par de registradores, denominados CCPR1L e CCPR1H.

O modo de captura pode ser configurado de quatro formas disponíveis:

- 1. Captura a cada borda de subida do sinal: Neste modo, o período do sinal é capturado a cada borda de subida (transição de “0” para “1”) do sinal aplicado ao pino CCP1(RB3).

- 2. Captura a cada borda de descida do sinal: Neste modo, o período do sinal é capturado a cada borda de descida (transição de “1” para “0”) do sinal aplicado ao pino CCP1(RB3).

- 3. Captura a cada 4ª borda de subida do sinal: Neste modo, o período do sinal é capturado a cada quatro bordas de subida do sinal aplicado ao pino CCP1(RB3). Isto significa que o sinal capturado será equivalente a quatro períodos do sinal de entrada. Isto permite uma maior precisão na medição do sinal.

- 4. Captura a cada 16ª borda de subida do sinal: Neste modo, o período do sinal é capturado a cada dezesseis bordas de subida do sinal aplicado ao pino CCP1(RB3). Isto significa que o sinal capturado será equivalente a dezesseis períodos do sinal de entrada. Isto permite uma precisão ainda maior que a do modo 3 na medição do sinal.

Cada vez que o sinal de entrada aplicado ao pino CCP1 gera uma borda (de subida ou descida, dependendo do modo de operação do módulo), é gerado um sinal de captura que copia a contagem atual do *timer1* (contador de 16 bits) no par de registradores CCPR1L e CCPR1H (8 bits em cada registrador). Além disso, o sinal de captura aciona também o *flag* de interrupção do módulo CCP1, fazendo com que gere uma interrupção do programa em execução [4].

O modo de captura foi utilizado porque o PIC 16F628A possui apenas uma interrupção externa, porém, apesar de não ser sua função principal, o modo de captura atende à necessidade do projeto, que é a geração de interrupção de borda, não sendo necessária a utilização dos registradores de captura.

Com as interrupções já definidas, é necessário ligar os pinos de saída dos comparadores com os pinos que geram as interrupções no programa. Uma característica importante do comparador que deve ser lembrada é que o mesmo possui uma impedância de saída baixa, o que exige a colocação de um resistor entre o pino de saída do comparador e a entrada do pino que gera a interrupção, para que o circuito funcione corretamente.

O programa executado no microcontrolador é baseado em interrupções, o que significa que o programa principal entra em um looping infinito e aguarda a execução das interrupções externas. A Figura 5.9 mostra o fluxograma do programa principal e os fluxogramas das interrupções externas:

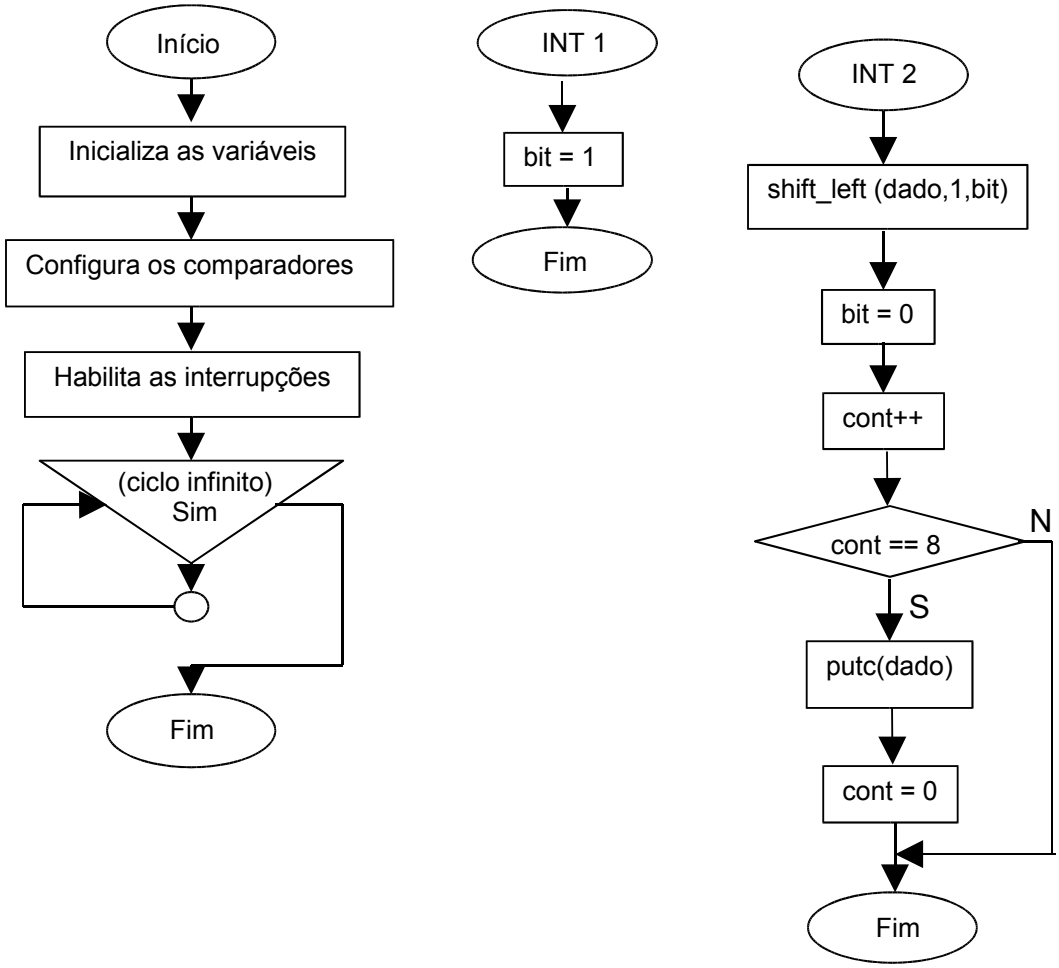


Figura 5.9 - Fluxograma do programa principal e das interrupções.

A função **shift_left** insere um bit em uma vatiável, matriz ou estrutura, deslocando à esquerda os demais bits dela.

Sintaxe:

bit = shift_left(e, n, v);

Em que:

- **e** é um ponteiro variável;
- **n** é um número inteiro ao tamanho em bytes da variável que será deslocada;
- **v** é uma variável ou constante booleana representando o valor do bit a ser inserido (0 ou 1).
- **bit** é uma variável booleana;

O bit v é inserido em direção ao bit mais significativo da variável apontada por **e**.

Exemplo:

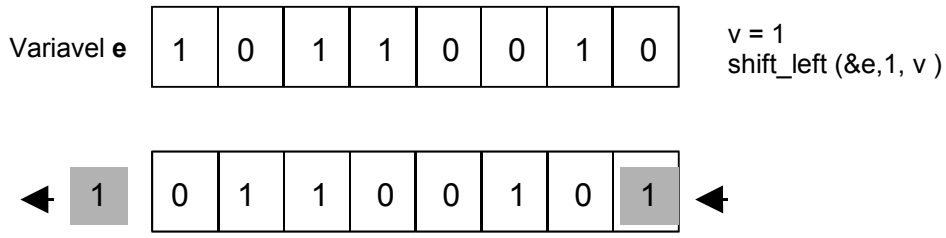


Figura 5.10 - Deslocamento de bits.

A função **putc**, simplesmente escreve um caractere na saída serial padrão. Esta função é utilizada para transmitir um dado ou caractere de 8 bits (dado) para o *Software Receptor*.

A interrupção INT 1 é configurada na borda de subida e a INT 2 na borda de descida. Dessa forma, quando as interrupções são acionadas pelo mesmo sinal da portadora, a interrupção INT 1 é sempre disparada primeira que a INT 2.

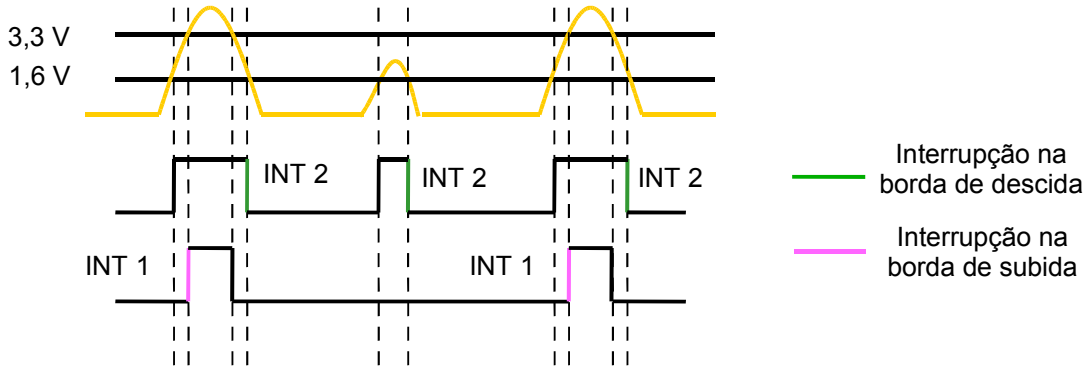


Figura 5.11 Borda de interrupção.

5.4 Módulo de Transmissão Serial

O microcontrolador possui uma USART interna, também chamada de SCI (Interface de Comunicação Serial), ele é um dispositivo utilizado para fazer a comunicação serial com elementos externos ao chip, tais como: computadores, modems, terminais, memórias, conversores A/D e D/A, etc.

O coração da USART é composto de dois registradores de deslocamento, responsáveis pela conversão paralelo/serial (transmissão), chamado internamente de TSR, e serial/paralelo (recepção), chamado internamente de RSR.

A transmissão serial assíncrona caracteriza-se pela ausência de uma linha de sincronização entre o elemento transmissor e o elemento receptor. Como não há sincronização de clock entre os elementos, utilizam-se velocidades padronizadas que devem ser seguidas por cada elemento de comunicação. Assim, uma vez que dois elementos estejam operando com uma mesma velocidade de comunicação, utilizam-se bits de START (início) e STOP (fim) para sinalizar o início e o fim de uma transmissão.

As rotinas de comunicação serial são muito afetadas pela imprecisão do clock, portanto, não é utilizado o clock interno do PIC 16F628A, pois o mesmo não possui uma calibração muito confiável e sua freqüência máxima de operação é de 4 Mhz, ao invés disso é utilizado um cristal externo de 20 Mhz.

O oscilador de cristal está contido num invólucro de metal com os pinos ligados aos pinos OSC1 e OSC2 do microcontrolador. Dois condensadores cerâmicos devem estar ligados a cada um dos pinos do cristal e ao terra:

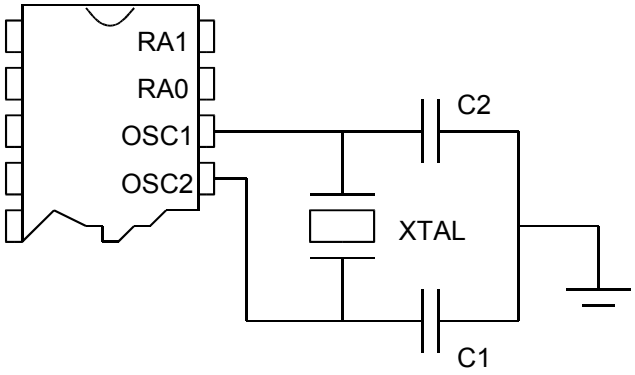


Figura 5.12 - Clock de um microcontrolador a partir de um cristal de quartzo

Quando se projeta um dispositivo, a regra é colocar o oscilador tão perto quanto possível do microcontrolador, de modo a evitar qualquer interferência nas linhas que ligam o oscilador ao microcontrolador.

Ao ligar a alimentação do circuito, o oscilador começa a oscilar. Primeiro com um período de oscilação e uma amplitude instável, mas, depois de algum tempo, tudo estabiliza.

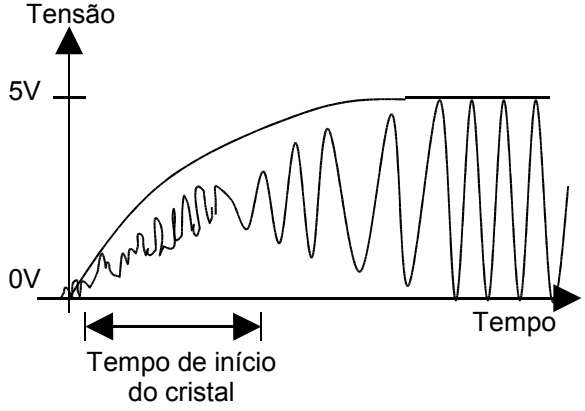


Figura 5.13 - Sinal de clock do oscilador depois de ser ligada a alimentação.

Assim como o microcontrolador, a maioria dos equipamentos digitais utilizam níveis TTL e CMOS. Portanto, o primeiro passo para conectar um equipamento digital a uma interface RS232 é transformar níveis TTL (0 a 5 volts) em RS232 e vice-versa. Isto é feito por conversores de nível. Existe uma variedade grande de equipamentos digitais que utilizam o *driver* 1488 (TTL => RS232) e o *receiver* 1489

(RS232 => TTL). Estes CIs contém 4 inversores de um mesmo tipo, sejam drivers ou receivers. O *driver* necessita de duas fontes de alimentação +7,5 volts a +15 volts e –7,5 volts a –15 volts. Isto é um problema onde somente uma fonte de +5 volts é utilizada. Um outro CI que está sendo largamente utilizado é o MAX232 (da Maxim). Ele inclui um circuito de “charge pump” capaz de gerar tensões de +10 volts e –10 volts a partir de uma fonte de alimentação simples de +5 volts, bastando para isso alguns capacitores externos, conforme pode-se observar na figura a seguir:

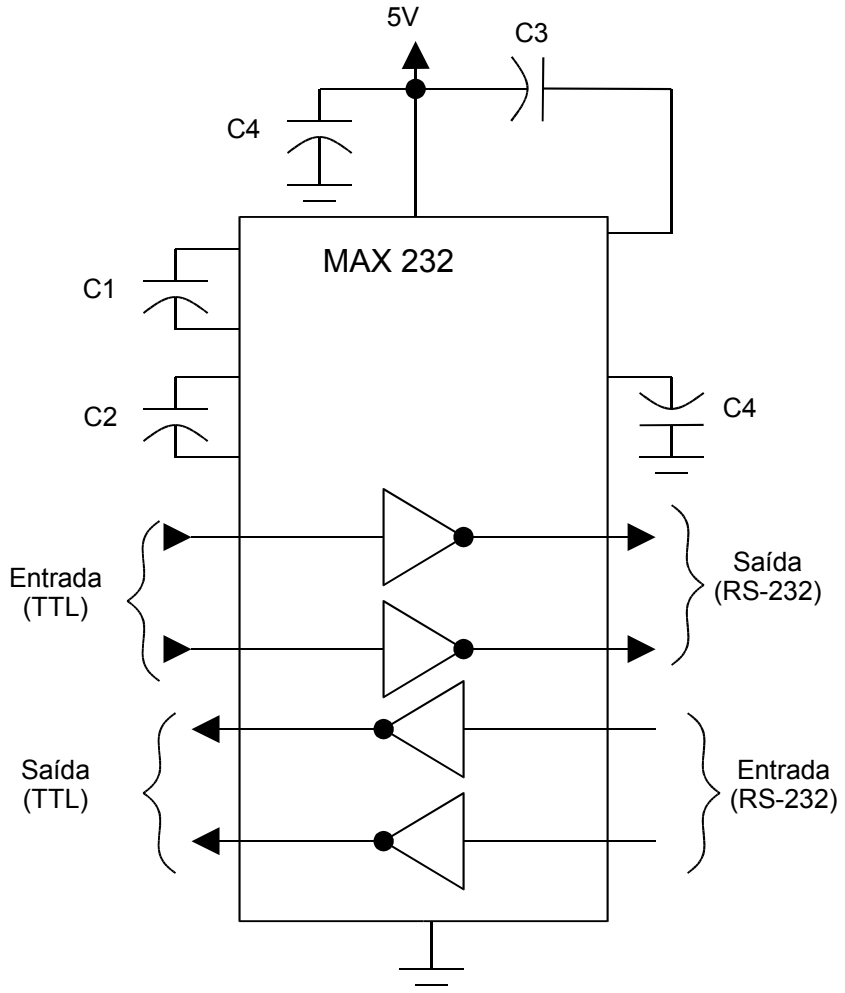


Figura 5.14 - Configuração do CI MAX232.

Este CI também tem 2 receivers e 2 drivers no mesmo encapsulamento. Nos casos onde serão implementados somente as linhas de transmissão e de recepção de dados, não seria necessário 2 chips e fontes de alimentação extras. Devido às essas características, o MAX232 foi escolhido para fazer a conversão do nível TTL na saída do microcontrolador para o nível de tensão aceito pelo padrão EIA232E.

5.5 Montagem do Protótipo

A Figura 5.1 mostra o circuito esquemático da interface completa, com a integração de todos os módulos: detecção, conversão A/D e transmissão serial.

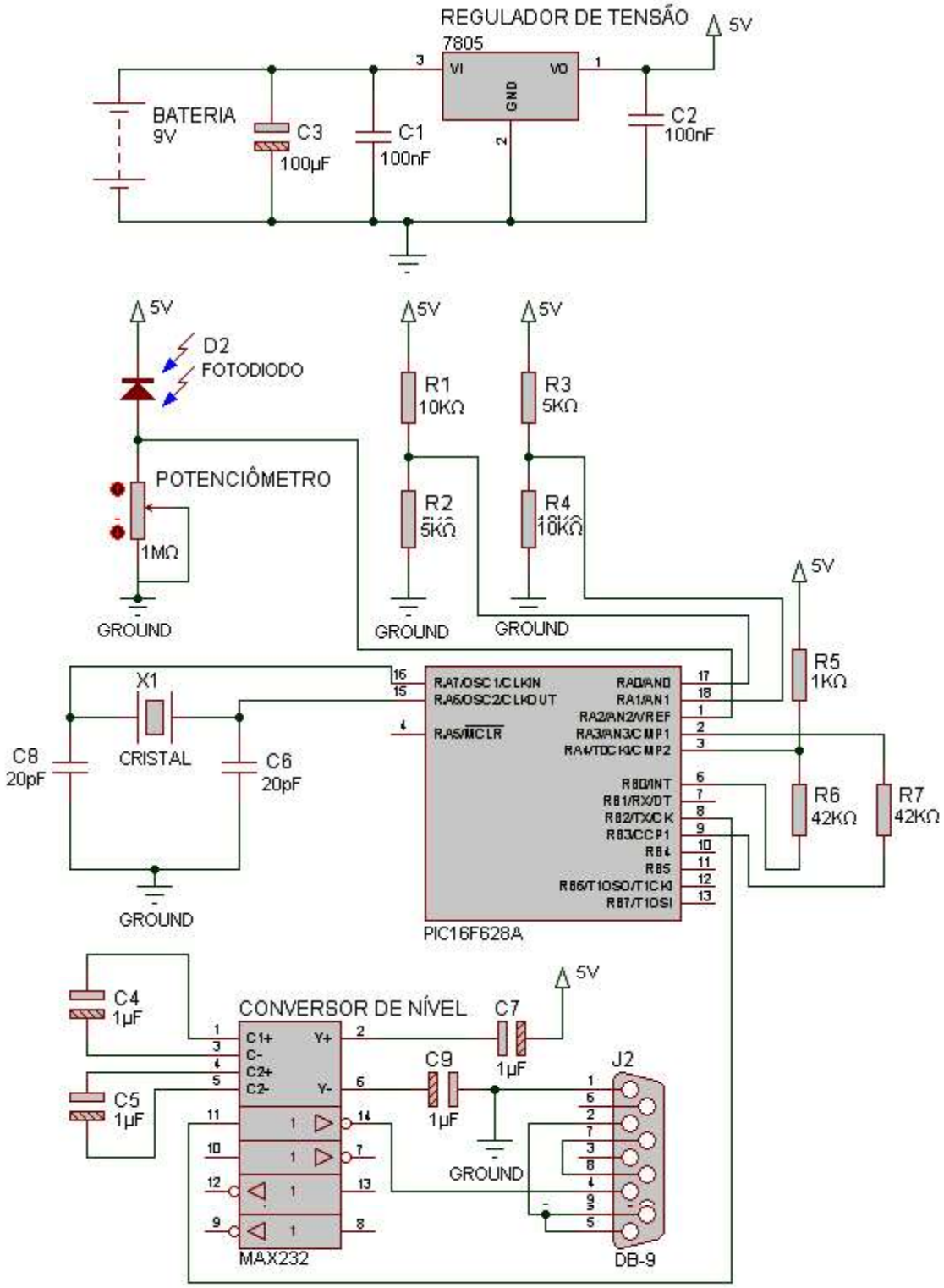


Figura 5.15 - Circuito esquemático.

As figuras abaixo mostram o protótipo completo. O circuito foi montado em uma placa de circuito impresso e protegido por uma caixa plástica.

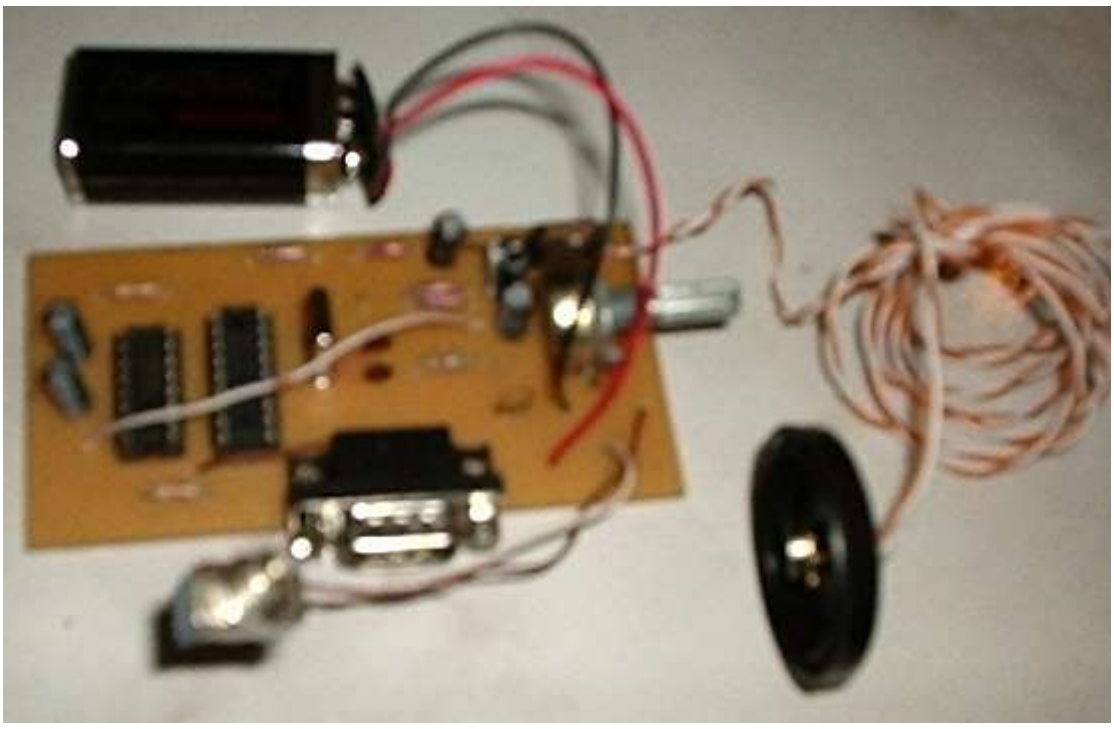


Figura 5.16 - Foto do protótipo 1.

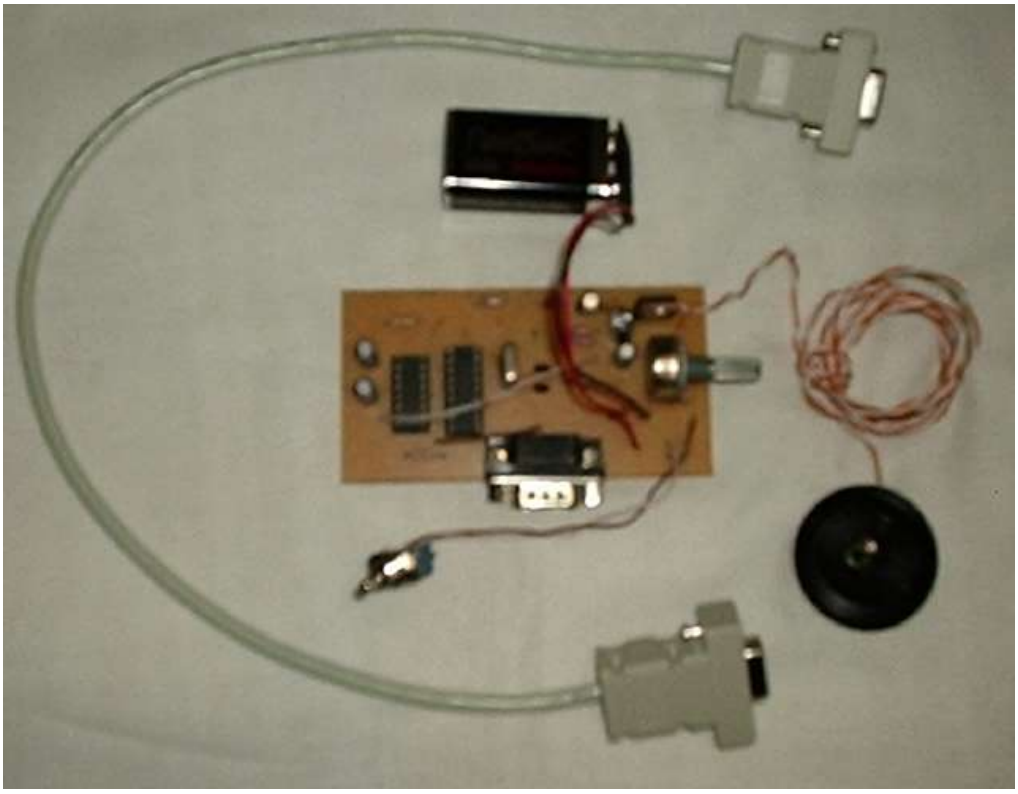


Figura 5.17 - Foto do protótipo 2.

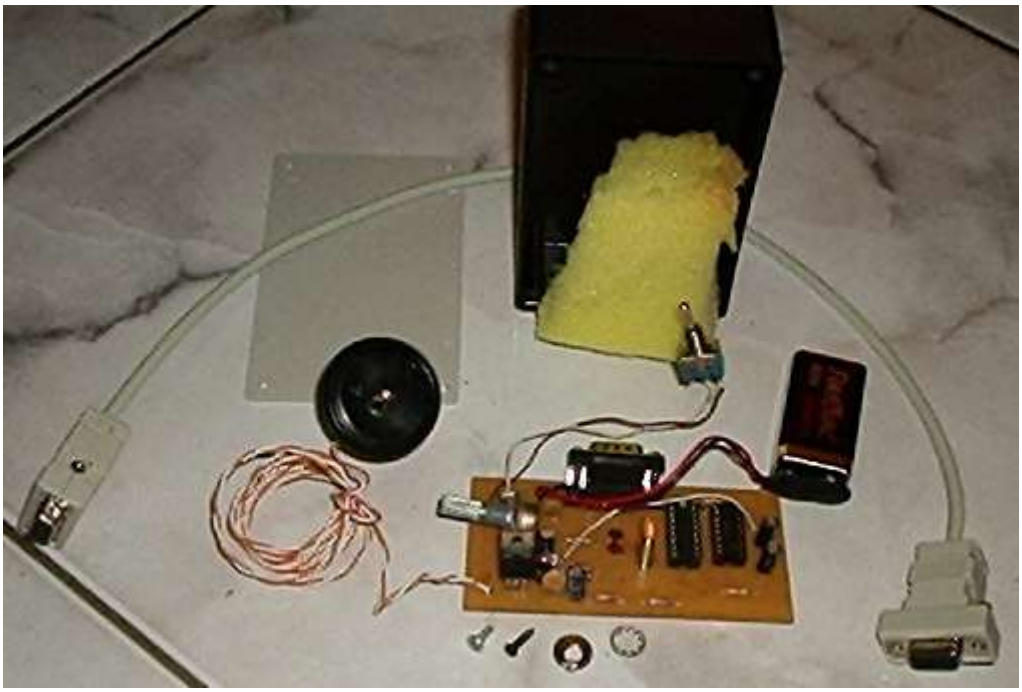


Figura 5.18 - Foto do protótipo 3.

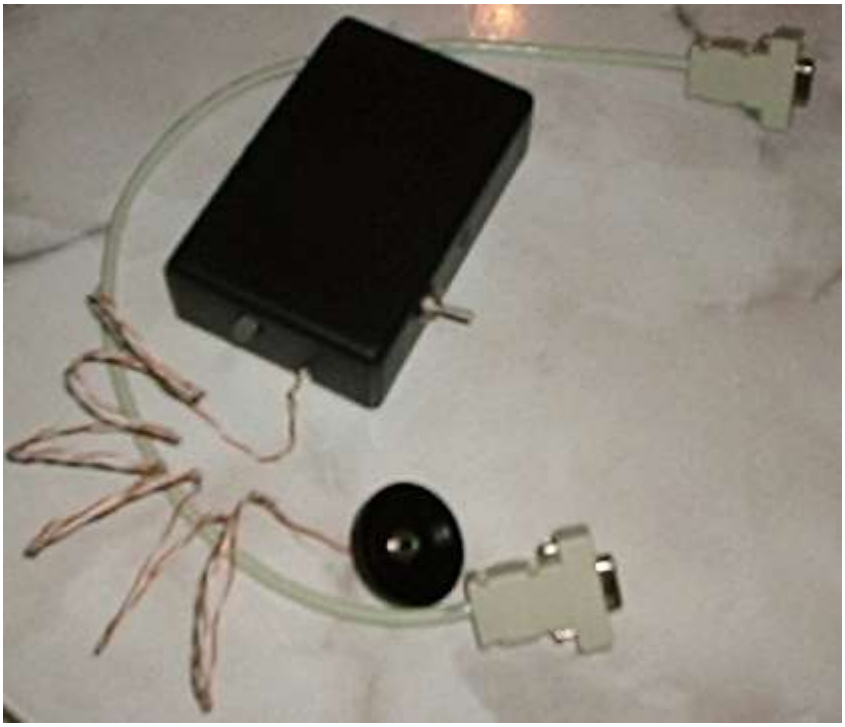


Figura 5.19 - Foto do protótipo 4.

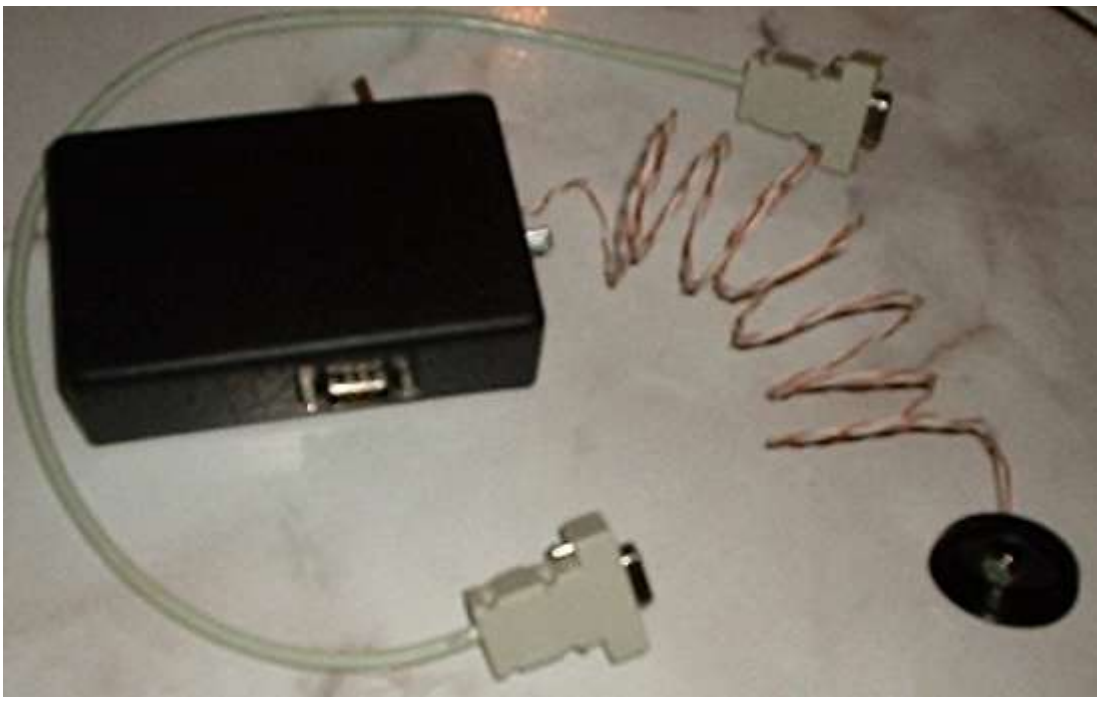


Figura 5.20 - Foto do protótipo 5.

6 CONSIDERAÇÕES FINAIS

6.1 Resultados Obtidos

Os primeiros testes, foram realizados com o *Software Transmissor* implementado totalmente em Java, utilizando a API *Full-Screen Exclusive Mode.* Estes testes não alcançaram resultados satisfatórios, pois a API não oferece garantia de sincronismo entre o redesenho realizado pelo *software* e a freqüência vertical do monitor.

A identificação desse problema exigiu um estudo em outras linguagens, técnicas de programação e API's. Durante a implementação do *Software Transmissor* vários testes foram realizados e a solução encontrada foi a utilização das linguagens Java e C++, utilizando a API Directx.

Portanto, o protótipo implementado funcionou de acordo com o esperado. Os sinais foram capturados da tela do monitor e transmitidos para o *software* do microcomputador receptor com sucesso. Porém, uma pequena falha foi identificada: os primeiros bits da transmissão sempre são capturados desordenados, danificando o primeiro caracter transmitido.

6.2 Conclusões

O uso do monitor como transmissor pode ser útil em ambientes onde as portas de comunicação do computador não são acessíveis ao usuário, porém, a velocidade de transmissão de dados por esse método se limita à freqüência vertical do monitor.

A velocidade de transmissão multi-níveis pode ser utilizada para aumentar a velocidade de transmissão, contudo, essa técnica é mais suscetível a erro de codificação por interferência.

6.3 Dificuldades encontradas

A principal dificuldade encontrada foi relacionada à implementação do *Software Transmissor*. A sua performance é fundamental para o funcionamento da transmissão, pois o mesmo tem que redesenhar as informações na tela na mesma

velocidade de atualização do monitor de vídeo.

Outro fato que dificultou a transmissão, é que se um bit for perdido durante a transmissão, todas as informações seguintes podem ser prejudicadas, pois os bytes se deslocam para a esquerda, embaralhando o restante da informação.

6.4 Sugestões para trabalhos futuros

A seguir, estão relacionadas algumas sugestões que podem melhorar a eficiência e a velocidade de transmissão:

- A interface de transmissão pode ser implementada operando com uma transmissão em multi-nível, de forma a aumentar a velocidade de transmissão;
- A interface de transmissão pode ser implementada utilizando mais de um fotodetector, operando com uma transmissão em paralelo;
- O protocolo de comunicação pode ser implementado com controle e detecção de erro entre os sinais ópticos transmitidos e o microcontrolador;
- A interface pode ser testada e avaliada nos diferentes tipos de monitores encontrados no mercado;
- O *Software Transmissor* pode ser implementado para funcionar na WEB, permitindo a transferência de arquivos de qualquer computador que possua acesso à Internet.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] - GIORDAN , M. **O Ensino de Ciências nos Tempos da Internet. em Ciência, Ética e Cultura na Educação**. Chassot e Oliveira (orgs), Ed. Unisinos, São Leopoldo, RS., 1998.

[2] - NASCIMENTO, Juarez do. **TELECOMUNICAÇÕES**. 2ª Edição. Makron Books, 2000.

[3] - OSBORNE, Adam. **Microprocessadores Conceitos Básicos**. Volume 1. McGraw-Hill, 1984.

[4] - PEREIRA, Fábio., **Microcontroladores PIC - Programação em C**. Editora Érica, 2004.

[5] - PEREIRA, Fábio., **Microcontroladores PIC - Técnicas Avançadas**. Editora Érica, 2004.

[6] - ROSCH, Winn L. **Desenvolvendo o Hardware do PC**. Campus LTDA., 1990.

[7] - VASCONCELOS, D. Gomes. **Comunicação Serial Utilizando a API da SUN**. Disponível em: <http://www.guj.com.br/content/articles/javacommapi/JavaCommAPI.pdf>. Acesso em: 21 nov. 2004).

[8] - VASCONCELOS, Laércio., **HARDWARE TOTAL**. MARKRON Books, 2002.

[9] - WHITE, Ron. **COMO FUNCIONA O COMPUTADOR III**. Quark, 1997.

ANEXO

ANEXO A - Comparador de tensão

Uma das aplicações do amplificador operacional é sua utilização como comparador de tensão, por isso, ambos possuem características semelhantes entra si. Um estudo prévio do amplificador operacional é necessário para melhor entendimento do funcionamento do comparador de tensão.

O amplificador operacional é referido como amp-op e é um circuito integrado onde quase todos os componentes eletrônicos necessários para operação estão empacotados num único encapsulamento.

Pode-se pensar no amp-op como um amplificador de propósito geral em diversas aplicações diferentes, especialmente em aplicações lineares. Sua principal aplicação é a reprodução de um sinal de entrada sem distorção e com uma amplitude muito maior, geralmente em condições de laço aberto (sem realimentação) os valores podem variar de 5.000 a 100.000.

O amp-op é esquematicamente representado como mostra a Figura A.1.

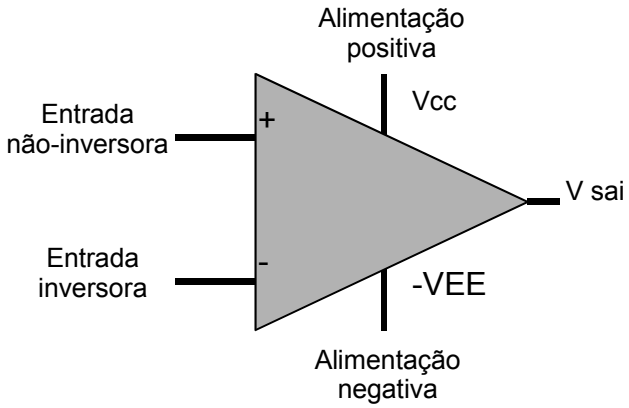


Figura A.1 - Símbolo esquemático do amplificador operacional.

Nota-se na figura acima que existem duas entradas do lado esquerdo que são denominadas de entrada inversora (indicada por um sinal de soma) e entrada não-inversora (indicada por um sinal de subtração). Ainda na Figura A.1 pode-se observar mais duas conexões, na parte superior é mostrado o terminal de alimentação positiva e na parte inferior o terminal de alimentação negativa.

Uma das aplicações do amplificador operacional é o seu uso como

amplificador diferencial. Isso significa que as diferenças de tensões entre as duas entradas mais o ganho de tensão do circuito do laço aberto resulta na tensão de saída amplificada.

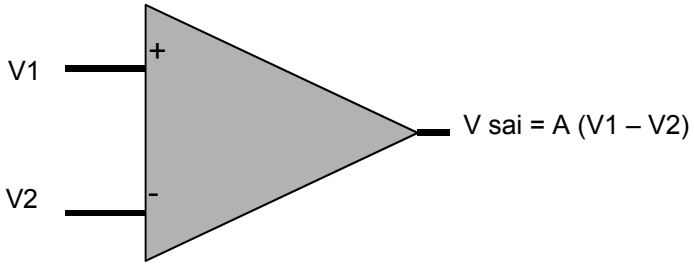


Figura A.2 - Amplificador diferencial, operação em laço aberto.

A característica de transferência do amp-op permanece linear somente sobre uma faixa limitada das tensões de entrada e saída. O amplificador opera com um limite de saída estabelecido entre a faixa de tensão da fonte de alimentação positiva e a fonte de alimentação negativa. Essa faixa de operação evita distorções na forma de onda de tensão de saída. A figura A.3 mostra a característica de transferência com os níveis de saturação positivo e negativo e as formas das ondas de entrada e saída.

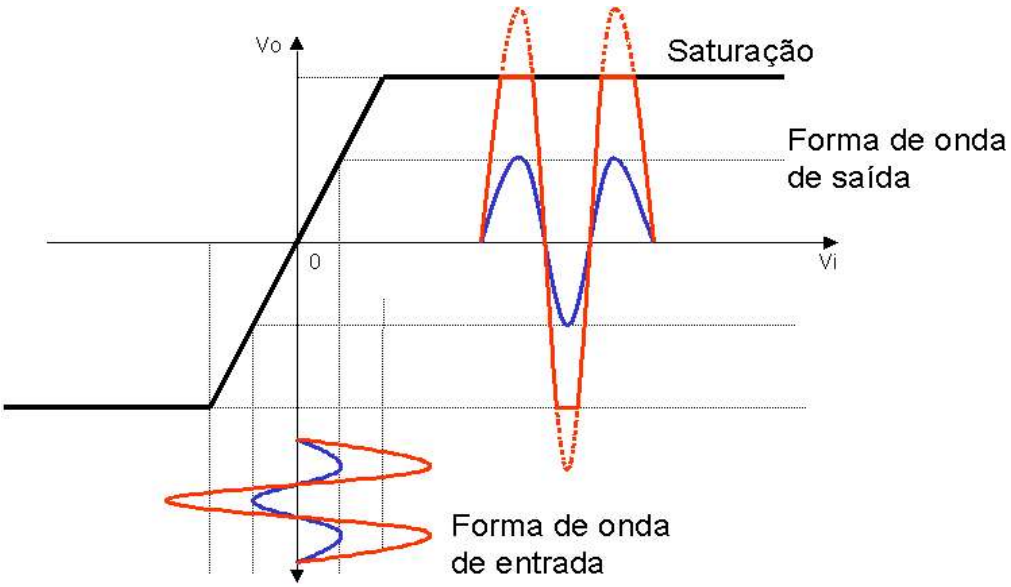


Figura A.3 - Característica de transferência, exceto onde ocorre a saturação.

Então, os níveis de tensão de alimentação especificam os limites de saída. Quando o amp-op está operando em saturação positiva ele pode atingir um máximo de aproximadamente 95% de Vcc (alimentação positiva) e quando operando em saturação negativa ele pode atingir 95% de Vee (alimentação negativa). Essa perda de 5% é devida a queda de tensão dentro do próprio amp-op, podendo atingir uma perda de até 2V.

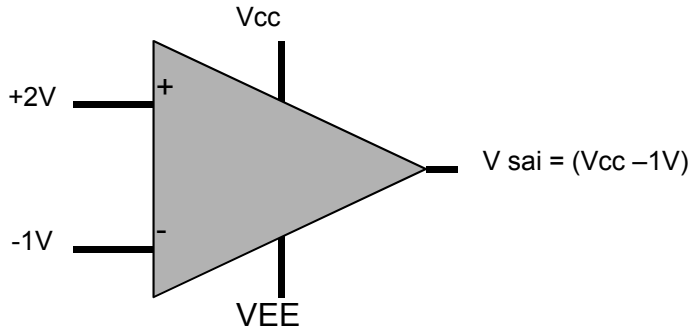


Figura A.4 - Amp-op saturando positivamente, operação em laço aberto.

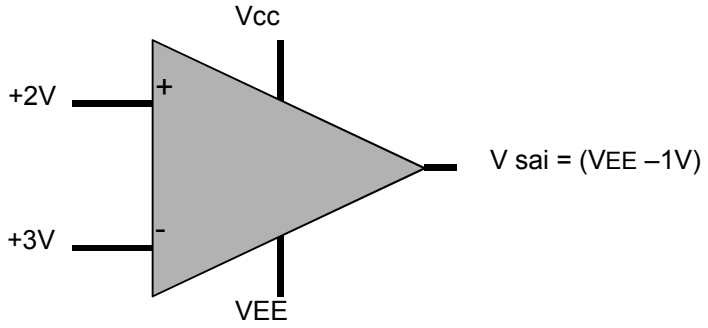


Figura A.5 - Amp-op saturando negativamente, operação em laço aberto.

Em muitos casos o amp-op é alimentado positivamente com uma tensão de +5V e a alimentação negativa é aterrada. O principal motivo dessa alimentação é que a variação da saída de 0V a +5V torna a saída do amp-op compatível com níveis de lógica transistor-transistor (TTL).

A Figura A.6 ilustra a operação do amp-op ideal com saída em níveis TTL e função transferência do amp-op.

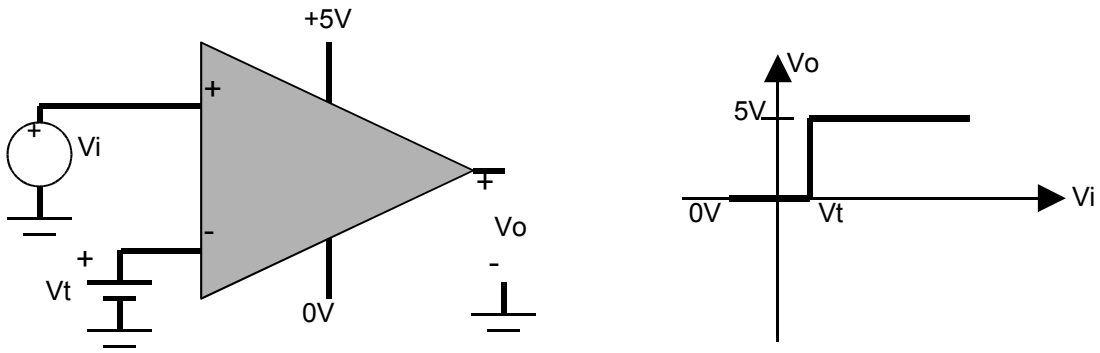
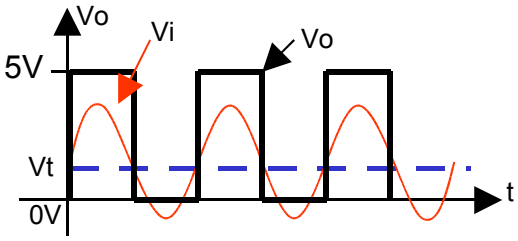


Figura A.6 - Operação ideal em níveis TTL e a função transferência de saída.



Se Vi>Vt, então Vo=5V

Figura A.7 - Função de transferência de entrada e saída.

Num circuito comparador, duas entradas diferentes são comparadas para ver qual das duas é maior. Se a entrada não-inversora é maior, a saída é forçada para saturação positiva. Quando a entrada inversora é maior, a saída do circuito é forçada para a saturação negativa.

É fácil notar como amp-op é usado para essa aplicação, porém, em muitos casos, dependendo do tipo de aplicação, a taxa de inclinação (*slew rate*) do amp-op limita a habilidade do circuito de responder rapidamente aos sinais de entrada.

Os comparadores são diferentes em dois aspectos significativos em relação aos amp-ops padrões:

- O primeiro: a saída faz parte de um coletor aberto. Num circuito com o coletor aberto, não há caminho de alimentação. Um resistor externo, conhecido como resistor de *pull-up* (se ligado a Vcc) ou *pull-down* (se ligado ao terra), deve ser conectado ao pino de saída para fornecer o caminho de alimentação para o CI.

- O segundo: o capacitor de compensação interna encontrado no lado de dentro de um amp-op está ausente. Esse capacitor de compensação interna tem uma função principal de ajudar evitar que oscilações indesejáveis apareçam. Ele faz isso reduzindo o ganho de tensão de laço aberto em altas freqüências, como parte de uma rede de atraso, projetado para desviar altas freqüências para o terra.

Note na Figura A.8 que o símbolo esquemático é semelhante ao usado pelo amp-op.

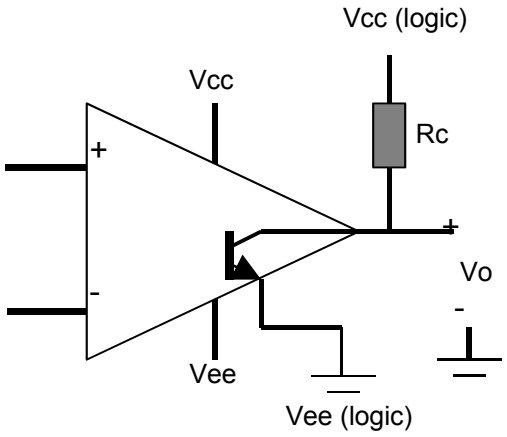


Figura A.8 - Característica do comparador de tensão.

ANEXO B - Descrição dos pinos - PIC 16F628A

Descrição dos pinos do microcontrolador PIC 16F628A:

Tabela B.1 - Descrição dos pinos.

Pino	Função	Tipo	Descrição
1	RA2/AN2/Vref	Entrada / saída	Porta A (bit 2) / entrada do comparador analógico / saída da referência de tensão.
2	RA3/AN3/CMP1	Entrada / saída	Porta A (bit 3) / entrada do comparador analógico / saída do comparador 1.
3	RA4/T0CKI/CMP2	Entrada / saída	Porta A (bit 4) / entrada de clock externo do timer 0 / saída do comparador 2.
4	RA5/MCLR/THV	Entrada	Porta A (bit 5) / reset CPU / tensão de programação.
5	Vss	Alimentação	Terra.
6	RB0/INT	Entrada / saída	Porta B (bit 0) / entrada de interrupção externa.
7	RB1/RX/DT	Entrada / saída	Porta B (bit 1) / recepção USART (modo assíncrono) / dados (modo síncrono).
8	RB2/TX/CK	Entrada / saída	Porta B (bit 2) / transmissão USART (modo assíncrono) / clock (modo síncrono).
9	RB3/CCP1	Entrada / saída	Porta B (bit 3) / entrada / saída do módulo CCP.
10	RB4/PGM	Entrada / saída	Porta B (bit 4) / entrada de programação LVP.
11	RB5	Entrada / saída	Porta B (bit 5).
12	RB6/T1OSO/T1CKI	Entrada / saída	Porta B (bit 6) / saída do oscilador TMR1 / entrada clock TMR1.
13	RB7/T1OSI	Entrada / saída	Porta B (bit 7) / entrada do oscilador TMR1.
14	Vdd	Alimentação	Alimentação positiva.
15	RA6/OSC2/CLKOUT	Entrada / saída	Porta A (bit 6) / entrada para o cristal oscilador / saída de clock.
16	RA7/OSC2/CLKIN	Entrada / saída	Porta A (bit 7) / entrada para o cristal oscilador / entrada de clock externo.
17	RA0/AN0	Entrada / saída	Porta A (bit 0) / entrada comparador analógico.
18	RA1/AN1	Entrada / saída	Porta A (bit 1) / entrada comparador analógico.

ANEXO C - Código Fonte - Software Transmissor

Programa Principal (implementado em Java):

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class TesteExec extends JFrame implements ActionListener
{
	JTextField rgb1, rgb2, rgb3;
	JPanel painelCor;
	String sR="0", sG="0", sB="0";
	JSlider slider;
	JButton btIniciar;
	JTextField texto;

public static void main(String []arg)
{
	TesteExec frame = new TesteExec();
	frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	frame.setVisible(true);
}

TesteExec()
{
	setTitle("Teste");
	setBounds(100,100,400,300);
	setResizable(false);
	JPanel painel = new JPanel();
	painel.setLayout(new BorderLayout());
```

```
        painelCor = new JPanel();
        painelCor.setBackground(Color.black);
        painel.add(painelCor,BorderLayout.CENTER);
        painel.add(getPainelControle(),BorderLayout.SOUTH);
        setContentPane(painel);
}

public JPanel getPainelControle(){
        JPanel painel = new JPanel();
        JPanel painel1 = new JPanel();
        painel.setLayout(new BorderLayout());
        painel1.add(new JLabel("R"));
        rgb1 = new JTextField("0",3);
        rgb1.addActionListener(this);
        painel1.add(rgb1);
        painel1.add(new JLabel("G"));
        rgb2 = new JTextField("0",3);
        rgb2.addActionListener(this);
        painel1.add(rgb2);
        painel1.add(new JLabel("B"));
        rgb3 = new JTextField("0",3);
        rgb3.addActionListener(this);
        painel1.add(rgb3);
        slider = new JSlider(0,255,0);
        slider.setMajorTickSpacing(10);
        slider.setMinorTickSpacing(1);
        slider.addChangeListener(new ChangeListener(){
                public void stateChanged(ChangeEvent arg0) {
                        int valor = slider.getValue();
                        painelCor.setBackground(new Color(valor,valor,valor));
                        rgb1.setText(""+valor);
                        rgb2.setText(""+valor);
                        rgb3.setText(""+valor);
                }
        });
        painel1.add(slider);
```

```
        painel.add(painel1,BorderLayout.NORTH);
        JPanel painel2 = new JPanel();
        painel2.setLayout(new FlowLayout());
        texto = new JTextField(25);
        painel2.add(texto);
        btIniciar = new JButton("Iniciar");
        btIniciar.addActionListener(this);
        painel2.add(btIniciar);
        painel.add(painel2,BorderLayout.SOUTH);
        return painel;
}

public void actionPerformed(ActionEvent evt) {
        if(evt.getSource() == btIniciar){
                try {
                        Process p = Runtime.getRuntime().exec("BasicDD.exe 255 "
                        +rgb1.getText()+" \""+texto.getText()+"\"");
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }else{
                try {
                        int R = Integer.parseInt(rgb1.getText());
                        if(R < 0 || R > 255){
                                JOptionPane.showMessageDialog(null,
                                "O valor deve ser entre 0 e 255");
                                R = Integer.parseInt(sR);
                                rgb1.setText(sR);
                        }else{
                                sR = rgb1.getText();
                        }
                        int G = Integer.parseInt(rgb2.getText());
                        if(G < 0 || G > 255){
                                JOptionPane.showMessageDialog(null,
                                "O valor deve ser entre 0 e 255");
                                G = Integer.parseInt(sG);
```

```
                    rgb2.setText(sG);
                }else{
                    sG = rgb2.getText();
                }
                int B = Integer.parseInt(rgb3.getText());
                if(B < 0 || B > 255){
                    JOptionPane.showMessageDialog(null,
                    "O valor deve ser entre 0 e 255");
                    B = Integer.parseInt(sB);
                    rgb3.setText(sB);
                }else{
                    sB = rgb3.getText();
                }
                painelCor.setBackground(new Color(R,G,B));
                } catch (NumberFormatException e) {
                    rgb1.setText(sR);
                    rgb2.setText(sG);
                    rgb3.setText(sB);
                }
            }
        }
}
```

Programa Secundário (implementado em C++):

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include <ddraw.h>
#include <iostream>
#include <cstdlib>

HWND                    g_hMainWnd;
HINSTANCE                   g_hInst;
LPDIRECTDRAW7           g_pDD = NULL;
```

```
LPDIRECTDRAWSURFACE7      g_pDDSFront = NULL;
LPDIRECTDRAWSURFACE7      g_pDDSBack = NULL;
BOOL teste = true;

/*---Prioridade--------*/
 HANDLE hProcess, hThread;
 DWORD ClassPriority;
 int ThreadPriority;

HWND InitWindow(int iCmdShow);
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam);
void ProcessIdle();
int InitDirectDraw();
void CleanUp();
void DDClear( LPDIRECTDRAWSURFACE7 pDDS);
void DDClearWhite( LPDIRECTDRAWSURFACE7 pDDS);
void DDClearBlack( LPDIRECTDRAWSURFACE7 pDDS);
DWORD CreateRGB( int r, int g, int b );
int c1;
int c2;
char* s;
bool *bits;
int bitLen;

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
          c1 = atoi(__argv[1]);
          c2 = atoi(__argv[2]);
          s = __argv[3];
          int len = strlen(s);
          int j = bitLen = len*8;
          bits = new bool[j];
```

```
	for(int i=--len; i>=0; i--){
		int dado = s[i];
		for(int k=0; k<8; k++){
			bits[--j] = (dado % 2 == 0) ? false : true;
			dado=dado/2;
		}
	}

	g_hInst = hInstance;
	g_hMainWnd = InitWindow(nCmdShow);
	if(!g_hMainWnd)
		return -1;
	if(InitDirectDraw() < 0)
	{
		CleanUp();
		MessageBox(g_hMainWnd, "Could start DirectX engine in your
computer. Make sure you have at least version 7 of DirectX installed.", "Error",
MB_OK | MB_ICONEXCLAMATION);
		return 0;
	}

	hProcess = GetCurrentProcess();
	hThread = GetCurrentThread();
	ClassPriority = GetPriorityClass(hProcess);
	ThreadPriority = GetThreadPriority(hThread);
	SetPriorityClass(hProcess, REALTIME_PRIORITY_CLASS);
	SetThreadPriority(hThread, THREAD_PRIORITY_TIME_CRITICAL);

	g_pDD->WaitForVerticalBlank(DDWAITVB_BLOCKEND, 0);
	g_pDD->WaitForVerticalBlank(DDWAITVB_BLOCKEND, 0);

	while( teste ) {
		MSG msg;
		if( PeekMessage( &msg, NULL, 0, 0, PM_REMOVE ) ) {
			if( msg.message == WM_QUIT )
```

```
                        break;

                    TranslateMessage( &msg );
                    DispatchMessage( &msg );
                }else{
                    ProcessIdle();
                }
            }

        SetPriorityClass(hProcess, ClassPriority);
        SetThreadPriority(hThread, ThreadPriority);
        CleanUp();

        delete [] bits;

        return 0;
}

void ProcessIdle()
{
        HRESULT hRet;
        static BOOL cor = false;
        static int pos=0;
        static cont=0;
        if(cor){
            if(bits[pos]){
                DDClearWhite( g_pDDSBack );
            }else{
                DDClear( g_pDDSBack );
            }
            if(++pos >= bitLen){
                cor=false;
                pos = 0;
            }
        }else{
            DDClearBlack( g_pDDSBack );
```

```
        }


        if(cont++ == 200){
                cor = true;
        }

        hRet = g_pDDSFront->Flip(NULL, DDFLIP_WAIT );
}

HWND InitWindow(int iCmdShow)
{

  HWND      hWnd;
  WNDCLASS  wc;
  wc.style = CS_HREDRAW | CS_VREDRAW;
  wc.lpfnWndProc = WndProc;
  wc.cbClsExtra = 0;
  wc.cbWndExtra = 0;
  wc.hInstance = g_hInst;
  wc.hIcon = LoadIcon(g_hInst, IDI_APPLICATION);
  wc.hCursor = LoadCursor(NULL, IDC_ARROW);
  wc.hbrBackground = (HBRUSH )GetStockObject(BLACK_BRUSH);
  wc.lpszMenuName = TEXT("");
  wc.lpszClassName = TEXT("Basic DD");
  RegisterClass(&wc);

  hWnd = CreateWindowEx(
         WS_EX_TOPMOST,
    TEXT("Basic DD"),
    TEXT("Basic DD"),
    WS_POPUP,
    0,
    0,
    GetSystemMetrics(SM_CXSCREEN),
    GetSystemMetrics(SM_CYSCREEN),
```

```
    NULL,
    NULL,
    g_hInst,
    NULL);

  ShowWindow(hWnd, iCmdShow);
  UpdateWindow(hWnd);
  SetFocus(hWnd);
  ShowCursor(false);

  return hWnd;

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
  switch (message)
  {
  case WM_KEYDOWN:
        if(wParam == VK_ESCAPE)
        {
              PostQuitMessage(0);
              return 0;
        }
        break;
  case WM_DESTROY:
    PostQuitMessage(0);
    return 0;

  }
  return DefWindowProc(hWnd, message, wParam, lParam);
}
int InitDirectDraw()
{
  DDSURFACEDESC2 ddsd;
```

```
  DDSCAPS2      ddscaps;
  HRESULT           hRet;

  hRet = DirectDrawCreateEx(NULL, (VOID**)&g_pDD, IID_IDirectDraw7, NULL);
  if( hRet != DD_OK )
     return -1;

  hRet = g_pDD->SetCooperativeLevel(g_hMainWnd, DDSCL_EXCLUSIVE |
DDSCL_FULLSCREEN );
  if( hRet != DD_OK )
     return -2;

  hRet = g_pDD->SetDisplayMode(640, 480, 16, 0, 0);
  if( hRet != DD_OK )
     return -3;

  ZeroMemory(&ddsd, sizeof(ddsd));
  ddsd.dwSize = sizeof(ddsd);
  ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
  ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE | DDSCAPS_FLIP |
DDSCAPS_COMPLEX;
  ddsd.dwBackBufferCount = 1;
  hRet = g_pDD->CreateSurface(&ddsd, &g_pDDSFront, NULL);
  if( hRet != DD_OK )
     return -1;
  ZeroMemory(&ddscaps, sizeof(ddscaps));
  ddscaps.dwCaps = DDSCAPS_BACKBUFFER;
  hRet = g_pDDSFront->GetAttachedSurface(&ddscaps, &g_pDDSBack);
  if( hRet != DD_OK )
     return -1;
  return 0;
}

void CleanUp()
{
        if(g_pDDSBack)
```

```
		g_pDDSBack->Release();

	if(g_pDDSFront)
		g_pDDSFront->Release();

	if(g_pDD)
		g_pDD->Release();
}
void DDClear( LPDIRECTDRAWSURFACE7 pDDS)
{
	HRESULT hr;
	DDBLTFX ddbfx;
	RECT    rcDest;
	if (pDDS == NULL)
		return;
	ddbfx.dwSize = sizeof( ddbfx );
	ddbfx.dwFillColor = CreateRGB(c2,c2,c2);
	rcDest.left=0;
	rcDest.right=640;
	rcDest.top=0;
	rcDest.bottom=480;
	hr = pDDS->Blt( &rcDest, NULL, NULL, DDBLT_WAIT |
DDBLT_COLORFILL , &ddbfx );
}

void DDClearWhite( LPDIRECTDRAWSURFACE7 pDDS )
{
	HRESULT hr;
	DDBLTFX ddbfx;
	RECT    rcDest;
	if (pDDS == NULL)
		return;
	ddbfx.dwSize = sizeof( ddbfx );
	ddbfx.dwFillColor = CreateRGB(c1,c1,c1);
	rcDest.left=0;
	rcDest.right=640;
```

```
        rcDest.top=0;
        rcDest.bottom=480;
        hr = pDDS->Blt( &rcDest, NULL, NULL, DDBLT_WAIT |
DDBLT_COLORFILL , &ddbfx );
 }

void DDClearBlack( LPDIRECTDRAWSURFACE7 pDDS )
{
        HRESULT hr;
        DDBLTFX ddbfx;
        RECT    rcDest;
        if (pDDS == NULL)
                return;
        ddbfx.dwSize = sizeof( ddbfx );
        ddbfx.dwFillColor = CreateRGB(100,100,100);
        rcDest.left=0;
        rcDest.right=640;
        rcDest.top=0;
        rcDest.bottom=480;
        hr = pDDS->Blt( &rcDest, NULL, NULL, DDBLT_WAIT |
DDBLT_COLORFILL , &ddbfx );
 }

DWORD CreateRGB( int r, int g, int b )
{
        return ((r/8)<<11) | ((g/4)<<5) | (b/8);
}
```

ANEXO D - Código Fonte - Software Receptor

Código fonte do Software Receptor (implementado em Java):

```
import java.io.IOException;
import java.io.InputStream;
import java.util.Enumeration;
import java.util.TooManyListenersException;
import javax.comm.CommPortIdentifier;
import javax.comm.PortInUseException;
import javax.comm.SerialPort;
import javax.comm.SerialPortEvent;
import javax.comm.SerialPortEventListener;
import javax.comm.UnsupportedCommOperationException;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SoftwareReceptor extends JFrame implements Runnable,
SerialPortEventListener, ActionListener{
	static CommPortIdentifier portId;
	static Enumeration portList;
	InputStream inputStream;
	static SerialPort serialPort;
	Thread readThread;
	JTextArea area;
	JButton btSair;
	JComboBox combo;
	String opcoes[]={"Caracter","Bits"};

public SoftwareReceptor(){
	try {
		serialPort = (SerialPort) portId.open("SoftwareReceptor", 2000);
	} catch (PortInUseException e) {
		JOptionPane.showMessageDialog(null,e.getMessage());
```

```
}
try {
        inputStream = serialPort.getInputStream();
} catch (IOException e) {
        JOptionPane.showMessageDialog(null,e.getMessage());
}
try {
        serialPort.addEventListener(this);
} catch (TooManyListenersException e) {
        JOptionPane.showMessageDialog(null,e.getMessage());
}
serialPort.notifyOnDataAvailable(true);
try {
        serialPort.setSerialPortParams(115200,
         SerialPort.DATABITS_8,
        SerialPort.STOPBITS_1,
        SerialPort.PARITY_NONE);
} catch (UnsupportedCommOperationException e) {}
readThread = new Thread(this);
readThread.start();
ExitWindow exit = new ExitWindow();
addWindowListener(exit);
setTitle("Software Receptor");
setBounds(100,100,400,400);
JPanel painel = new JPanel(new BorderLayout());
area = new JTextArea();
area.setLineWrap(true);
JScrollPane scroll = new JScrollPane(area);
painel.add(scroll,"Center");
JPanel btPainel = new JPanel(new FlowLayout());
btSair = new JButton("Limpar");
btSair.addActionListener(this);
btPainel.add(btSair);
combo = new JComboBox(opcoes);
btPainel.add(combo);
painel.add(btPainel,"South");
```

```
        setContentPane(painel);
        show();
}

public void run() {
        try {
                Thread.sleep(20000);
        } catch (InterruptedException e) {}
}

public void serialEvent(SerialPortEvent event) {
        switch(event.getEventType()) {
                case SerialPortEvent.BI:
                case SerialPortEvent.OE:
                case SerialPortEvent.FE:
                case SerialPortEvent.PE:
                case SerialPortEvent.CD:
                case SerialPortEvent.CTS:
                case SerialPortEvent.DSR:
                case SerialPortEvent.RI:
                case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
                break;
                case SerialPortEvent.DATA_AVAILABLE:
                readData();
                break;
        }
}

public static void main(String []args)
{
        Object[] possiveisValores = {"COM1","COM2","COM3","COM4"};
        String valorSelecionado = (String) JOptionPane.showInputDialog(null,"Porta",
        "Selecione a porta", JOptionPane.INFORMATION_MESSAGE, null,
        possiveisValores, possiveisValores[0]);
        if(valorSelecionado != null && !valorSelecionado.equals("")){
                portList = CommPortIdentifier.getPortIdentifiers();
```

```
                while (portList.hasMoreElements()) {
                        portId = (CommPortIdentifier) portList.nextElement();
                        if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                                if (portId.getName().equals(valorSelecionado)) {
                                SoftwareReceptor frame = new SoftwareReceptor();
                                }
                        }
                }
                if(serialPort == null){
                        JOptionPane.showMessageDialog(null,"Porta não existe");
                        System.exit(0);
                }
        }else{
                System.exit(0);
        }
}

class ExitWindow extends WindowAdapter{
        public void windowClosing(WindowEvent e){
                System.exit(0);
        }
}

public void actionPerformed(ActionEvent e) {
        area.setText("");
}

public void readData()
{
        String str;
        Int bytes;
        byte[] readBuffer = new byte[2048];
        try{
                while (inputStream.available() > 0 ){
                        bytes = inputStream.read(readBuffer);
                        if (bytes > 0){
```

```
                    if (bytes > readBuffer.length)
                    {
                            System.out.println(serialPort.getName()+
                             ": Input buffer overflow!");
                    }
                    if(combo.getSelectedIndex() == 0)
                            str =  displayText(readBuffer, bytes);
                    else
                            str = funcaoBit(readBuffer, bytes);
                    }
            }
        }
        catch (IOException ex)
        {
            System.out.println(serialPort.getName()+ ": Cannot read input stream");
        }
}

private String displayText(byte[]    bytes, int       byteCount)
{
        String str;
        Int i, idx;
        byte[] nb;
        nb = new byte[byteCount * 4];
        for (i = 0, idx = 0; i < byteCount; i++){
            if (Character.isISOControl((char) bytes[i])
            && !Character.isWhitespace((char) bytes[i])){
                    nb[idx++] = (byte) '<';
                    nb[idx++] = (byte) '^';
                    nb[idx++] = (byte) (bytes[i] + 64);
                    nb[idx++] = (byte) '>';
            }else{
                    nb[idx++] = bytes[i];
            }
        }
        str = new String(nb, 0, idx);
```

```
        area.append(str);
        return(str);
}

 public String funcaoBit(byte[] dado,int    byteCount){
        String str="";
        StringBuffer b = new StringBuffer();
        for(int i=0; i<byteCount; i++){
                String c = Integer.toBinaryString(dado[i]);
                b.append(c);
        }
        area.append(b.toString());
        area.append(str);
        area.setCaretPosition(area.getText().length());
        return(str);
}
}
```

ANEXO E - Código Fonte - Microcontrolador PIC 16F628A

Código fonte do programa executado no microcontrolador PIC (implementado em C):

```
#include <16f628A.h>
#use delay(clock=20000000)
#fuses HS, NOWDT, PUT, NOBROWNOUT, NOLVP, NOMCLR
#use rs232 (baud=115200, xmit=PIN_B2,rcv=PIN_B1)
int8 i=0;
boolean bit=0;
byte dado;

#INT_EXT
void subida(){
  bit = 1;
}

#INT_CCP1
void descida(){
        shift_left(&dado,1,bit);
        bit=0;
        if(++i == 8){
                i=0;
                putc(dado);
        }
}

main(){
        setup_comparator(A0_A2_A1_A2_OUT_ON_A3_A4);
        setup_ccp1(CCP_CAPTURE_FE);
        setup_timer_1(T1_INTERNAL);
        EXT_INT_EDGE(L_TO_H);
        enable_interrupts(INT_CCP1);
        enable_interrupts(INT_EXT);
```

```
        enable_interrupts(GLOBAL);
        printf("\n\rIniciando transmissão...\n\r");
        delay_ms(200);
        bit=0;
        i=0;
        dado=0;
        while(true);
}
```